Modeling and prediction of thermal dissipation in heterogeneous CPU platforms

---

Joel Öhrling
1901937

Master Thesis in Computer Engineering
Supervisors: Dragos Truscan, Sébastien Lafond
Faculty of Science and Engineering
Åbo Akademi University
2020

Abstract

For 5G, the next generation of telecommunications technology, energy efficiency is an important aspect. It has been shown that cooling and heat management accounts for a large part of the energy consumed by the telecommunications infrastructure. Therefore, there is a need to address how heat dissipation can be limited and the energy spent on cooling reduced. This thesis explores thermal modeling approaches for multi-core heterogeneous processors and investigates which approaches can be utilized to predict the thermal dissipation with the highest accuracy. Previously proposed approaches to thermal modeling and system identification have been investigated and reviewed. Based on the review, three approaches were selected for implementation and compared in terms of prediction accuracy: a linear state-space identification approach using polynomial regressors, a NARX neural network approach and a recurrent neural network approach configured in an FIR model structure. These modeling approaches were each assessed for both 1 and 6 hours of training data collected from a multi-core heterogeneous ARM processor. The results showed that the state-space model based on polynomial regressors significantly outperformed the other two modeling approaches when trained with 1 hour of data. When the models were trained with 6 hours of data, all three modeling approaches yielded good results. However, the state-space approach still produced the lowest prediction error of the approaches.

Keywords: System identification, Thermal modeling, ARM, heterogeneous processor, N4SID, NARX, GRU

Preface

This thesis is written as a final thesis project in the Nordic Master Programme in Intelligent Software Systems (NISS). It was conducted as a part of the MegaM@Rt research project, a European industry-academia collaboration that aims to create model-based tools for continuous development and validation of runtime systems. I would like to thank everyone involved in the project and everyone at the Embedded System Labs at Åbo Akademi University. A special thank you to my supervisors Dragos Truscan and Sébastien Lafond as well as Wictor Lund at the ES Labs. I would also like to thank everyone at Datateknologerna r.f vid Åbo Akademi for the tremendous support and a 'kiva' time during my stay in Finland.

Contents

List of Figures

List of Tables

1 Introduction

In an age of ever-increasing demand for mobile data traffic and the number of Internet-connected devices growing every day, new technologies to meet these demands on throughput and availability are continuously being researched and deployed. With ever more dire climate reports presented every year, it is also crucial to consider the climate impact of these telecommunications systems. For the coming generation of telecommunications technology, 5G, energy efficiency is, therefore, an essential component. In 2016, it was estimated that around 5% of the world's CO_2 emissions originated from information and communication technology [1]. The same year YouTube alone contributed with 10 million tons of CO_2-equivalent emissions [2], roughly twice the annual carbon footprint of the Helsinki Metropolitan Area [3]. These numbers are expected to increase as progressively more users and services are connecting to the internet every day. The GSM Association has predicted that there will be over 1.8 billion users connected to 5G networks in 2025 [4]. Furthermore, according to an investigation by Andrae et al., the worst-case estimation on the percentage of greenhouse gas emissions caused by information and communication technology globally in the year 2030 is 23% [5]. Thus, there is an urgent need to focus more research on the energy efficiency and power dissipation of communications technology.

A large portion of the energy consumed by cellular networks is consumed by the infrastructure itself. An investigation into the power consumption of cellular networks by Alsharif et al. [6] revealed that base stations account for more than half of the total consumed energy. Alsharif et al. also found that cooling and heat management account for 10-25% of a typical base station's power consumption. However, an article by Tu et al. [7] suggests that this number could be as high as 50%. Thus, finding ways to reduce thermal dissipation in base stations and limit the amount of cooling required is an essential factor in reducing the overall climate impact of the wireless communications infrastructure.

1.1 Heterogeneous platforms

The data processed by base stations have a high degree of paralellizability. Therefore, heterogeneous computing techniques are being deployed to enable processing of multiple calls, packets, and data streams at once as well as handling signal processing tasks such as modulation and coding [8]. Heterogeneity in computer systems refers to computers that integrate parts that have different architectures or are optimized to perform different types of tasks. A modern PC is an example of a heterogeneous computer. It generally features both a central processor and several other types of processors that deal with peripherals, such as GPUs, sound cards or network cards. For many years, computers relied on very centralized architectures with often a single core executing all instructions. Over the past 15 years, however, there has been a shift towards increasing the number of cores instead of the clock frequency [9].

Today, there exists a wide range of chips that integrate several types of components into the same casing. These are referred to as a System-on-a-Chip (SoC). An SoC is typically a fully functioning computer on a single chip. Components that have traditionally been discrete parts in a large computing system are now being integrated on the same chip. For heterogeneous systems, this means that multiple types of processing units are integrated into the same chip. These types of chips are generally referred to as heterogeneous SoCs or heterogeneous processors. Heterogeneous processors can, for example, be found in most of today's mobile phones and are becoming progressively more prevalent in all types of computers [10]. This is also the case for information and communications systems [8]. The primary motivation for deploying heterogeneous processors in these systems is to provide better efficiency, as different types of cores are optimized for different types of instructions or workloads [11].

There are a large number of architectures for heterogeneous processors. A heterogeneous system architecture (HSA) is a standardized specification that allows for the integration of different types of processors on the same bus. Architectures that combine two or more multi-core CPUs are conventional in many energy-constrained devices today. The purpose of these architectures is to provide better energy efficiency by combining cores that have different power consumption characteristics and performance. An example of an architecture with multiple CPUs is the big.LITTLE architecture from the British processor design company ARM [12]. It combines a cluster of high-performance cores with a cluster of cores with better energy efficiency. This enables the platform to provide high throughput while still being able

to run energy efficiently when throughput is low. This type of architecture with a set of high-performance cores and a set of energy-efficient cores is today commonplace in most mobile phones [10]. This thesis will focus on this multi-CPU type of heterogeneity and the big.LITTLE architecture.

1.2 Modeling of heterogeneous systems

Testing and verification of the thermal characteristics of a heterogeneous multi-core processor can be an exhaustive process with many parameters to consider. Even for a single-core processor, there is a large number of variables that impact how much heat a processor dissipates, for example, the ambient temperature, the workload application and the core frequency. For heterogeneous multi-core platforms, this number grows even further as each processing element comes with its own unique characteristics. Performing exhaustive testing on these types of systems becomes unfeasible, as the number of possible configurations is vast. Constructing models of a system is, therefore, an alternative that can be considered. Building a model can enable the prediction of thermal dissipation in a system and can help speed up the test and verification process.

The area of system identification is a research domain that deals with approaches to creating mathematical models of dynamical systems through statistical and machine learning approaches. System identification also deals with how to model a system using limited amounts of data.

In system identification, modeling approaches are usually categorized into one of three groups: white-box modeling, gray-box modeling and black-box modeling. White-box modeling is an entirely theoretical modeling approach, where the model is constructed based on knowledge about the system and does not rely on data. White-box approaches require a complete knowledge of all the properties, parameters and uncertainties of a system. Black-box modeling is the complete contrast to white-box modeling. Black-box approaches depend solely on observations of a system without relying on insights into the underlying process. Data-driven modeling is another name to describe black-box modeling. The third group of system identification techniques is gray-box modeling. Gray-box techniques rely on parts of both black- and white-box modeling. Figure 1.1 shows the three modeling classes and how they relate to each other.

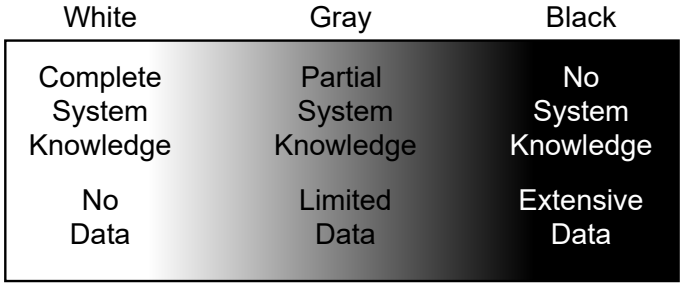

Figure 1.1: Illustration of the color-based categorization of modeling approaches.

In reality, as most models are constructed based on some knowledge and observations of a system, most modeling approaches can be viewed as being gray-box approaches to some degree [13].

1.3 Goal and purpose

This thesis project aims to investigate system identification approaches that can be utilized to estimate the heat produced by a multi-core heterogeneous processor. Multi-core and heterogeneous systems are becoming more and more common in all types of applications. Thus, the focus in this thesis will be on these types of CPUs and, more specifically, on heterogeneous CPUs utilizing the ARM big.LITTLE architecture. Additionally, since the data processed by cellular base stations is commonly highly parallel data streams, this thesis will focus on this class of applications.

This thesis will review existing approaches to modeling of thermal dissipation patterns in processors and computing systems as well as study the state-of-the-art in the field of system identification. The ultimate goal of this thesis is to assess the performance of a few system identification approaches for thermal modeling of heterogeneous multi-core processors in a comparative study.

1.4 Limitations

A limitation for this thesis is to only explore heterogeneous processors in the context of highly parallel applications, i.e., applications that can be substantially sped up by running on multiple cores. Moreover, a common approach to thermal modeling is to utilize power consumption measurements from the processor and exploit the relationship between power and thermal dissipation. Heterogeneous processors are, however, not commonly equipped with sensors measuring the power consumption

for each individual core. Therefore, this thesis will consider modeling approaches that strictly rely on the core frequencies and each core's utilization percentage to make predictions about thermal dissipation. Other factors also impact the heat dissipation and heat transfer of a processor. This thesis will not consider the impact of varying ambient temperature and humidity as well as active cooling.

1.5 Problem formulation

Based on the goals and problem domain mentioned above, the main research question of this project is:

> Utilizing data collected from a number of random configurations of a heterogeneous processor over a limited amount of time, which approaches to modeling can produce the most accurate model in terms of prediction accuracy?

On the basis of this goal, two research questions are posed:

> ***RQ1:*** *What modeling approaches exist for prediction of heat dissipation in multi-core heterogeneous processors?*
> ***RQ2:*** *Is there a difference in prediction accuracy between data-driven black-box approaches and gray-box model-based approaches when applied to predict the temperature of a multi-core heterogeneous processor?*

To address the first research question, a review of the state-of-the-art will be conducted. Previous research in the areas of thermal modeling of processors and non-linear system identification will be investigated. The outcome of answering this research question serves as a basis for addressing the second research question.

The second question intends to identify which types of modeling approaches produce the lowest prediction error. A few modeling approaches will be selected based on the result of *RQ1*. They will be assessed in a comparative study that tests the modeling approaches performance when trained with two different lengths of training data. Thus, *RQ2* can be split into two sub-questions:

> ***RQ2-1:*** *Which modeling approach achieves the lowest prediction error when trained with 1 hour of data?*
> ***RQ2-2:*** *Which modeling approach achieves the lowest prediction error when trained with 6 hours of data?*

1.6 Thesis structure

This thesis is divided into several chapters. Chapter 2 describes some theory behind power and thermal dissipation in processors as well as sheds light on common approaches to power and thermal management. The following chapter details theory behind modeling, system identification and machine learning concepts commonly applied to tackle these problems. In Chapter 4, the state-of-the-art in modeling of computer systems and general trends in the system identification domain is presented. Chapters 5 describes the methodologies, experimental setup and validation procedures applied in this study. Chapter 6 contains a more thorough description of the modeling approaches selected for the comparative study. Chapter 7 contains a presentation of the results achieved and an evaluation of the results of the study. In the following chapter, discussion and reflection on the outcome of the study are presented. The final chapter presents some concluding remarks and a few potential improvements and future possibilities for this present work.

2 Power and heat dissipation in CPUs

Although heterogeneous multi-core CPUs introduce many new possibilities for the power and thermal management of computers, the same principles that dictate the power and heat dissipation of a single-core CPU still applies. This chapter will describe the theory behind the power and heat dissipation of a processor and how the heat dissipation in a processor can be modeled. This chapter will also introduce a few common techniques that processors utilize to manage heat and power consumption.

2.1 Power dissipation

The power that a processor dissipates originates from the large number of transistors that are switching as the CPU executes instructions. The total power consumption of a CPU core can be described as the sum of three types of power consumption: the dynamic power consumption, the static power consumption and the short-circuit power consumption. The short-circuit power has, however, been shown to be practically negligible in modern CPUs [14]. Therefore, Equation (2.1) is the formula commonly used to describe the power consumption, where P is the total power, P_{dyn} is the dynamic power and P_{sta} is the static power.

$$P = P_{dyn} + P_{sta} \tag{2.1}$$

The dynamic power is the rate at which energy is consumed within the logic gates of a CPU core when an instruction is being executed. This power is proportional to the core frequency and the core voltage squared. Equation (2.2) shows the relationship between the dynamic power consumption P_{dyn}, the total switching capacitance C, the core voltage V_{dd} and the clock frequency f. The factor α represents the proportion of transistors that are activated during a given clock cycle.

$$P_{dyn} = \alpha \; C \; V_{dd}^2 \; f \tag{2.2}$$

Static power is dissipated at all times a processor is powered on and constitutes of leakage currents in the transistors. This power is, unlike the dynamic consumption, not directly dependent on the core frequency. Even when the transistors are in

an idle state and not supposed to draw any power, they still draw a small leakage current. Considering that there are more than a billion transistors in a modern CPU [15], these small leakages of current add up to a significant amount of power. In Equation (2.3), the formula for the static power consumption is shown. Here, L is the transistor count, V_{dd} is the core voltage and I_{leak} is the leakage current for each transistor.

$$P_{sta} = L\ V_{dd}\ I_{leak} \tag{2.3}$$

The leakage current is a combination of several types of leakage in the transistors. The primary sources are the sub-threshold leakage current and the gate-oxide leakage current [16]. The sub-threshold current is leakage in between the source and the drain when the transistor is in an off-state. The sub-threshold current is dependent on three voltages: the supply voltage, the threshold voltage, and a thermal voltage. The threshold voltage for a CMOS transistor is the voltage that is required for the transistor to start conducting current into the drain. The thermal voltage is temperature-dependent and is linearly proportional to the core temperature [16]. The gate oxide is the layer that separates the transistor gate from the source and the drain. The thinner this layer is, the more leakage current will seep through.

Historically, the static power consumption has been several orders of magnitude lower than the dynamic power consumption [17]. However, as the transistor size has decreased, the leakage current has increased [18]. Both the gate oxide leakage current and the sub-threshold leakage current have been shown to increase as the processor technology becomes smaller [19]. The static power contributes, therefore, a significant portion of the total power consumption in today's processors.

As the leakage current is dependent on the temperature, the CPU core temperature will affect how much power it dissipates. It has been shown in a study by Kocanda et al. [20] that the static power consumption in CMOS gates increases as the temperature increases. This is due to the sub-threshold leakage current in the transistors being dependent on the temperature. The static power consumption is, therefore, a feedback loop, which means that the static power and the core temperature are both positively correlated with one another. In [21], De Vogeleer et al. showed that the total power consumption of a Cortex-A15 processor increased by approximately 20% between 25°C and 85°C. Furthermore, Holmbacka et al. [22] demonstrated that the static power consumption of an Cortex-A9 processor more than doubled between 1°C and 80°C.

2.2 Power and thermal management

To manage power consumption and maintain its operation within thermal limits, a processor is equipped with several tools to control how much energy that is consumed and subsequently how much temperature that is dissipated. These tools can be both in the form of software deployed at kernel level or hardware support implemented in the processors. This section will present a few of these frequently applied techniques.

2.2.1 Dynamic voltage and frequency scaling

A technique that is commonplace in most processors today is Dynamic Voltage and Frequency Scaling (DVFS). It is a technique that enables the management of power consumption by taking advantage of the broad operating region of the CMOS transistor [23]. As the energy consumed by a CMOS transistor is dependent on the voltage squared, lowering the voltage can reduce the power consumed by the CPU. Especially, the dynamic component of the power consumption can be reduced by this technique. The formulas given in Section 2.1, show that the dynamic power consumption is also proportional to the square of the core voltage. The switching speed of a CMOS gate is, however, linearly dependent on the voltage. Thus, lowering the voltage will reduce the power consumption exponentially while the performance is only reduced linearly [24].

The frequency can also be adjusted in most modern processors. In theory, the frequency can be scaled to any arbitrary value. However, the frequency must, in practice, adhere to a specific range that is dictated by the switching speed and the threshold voltage of the CMOS transistors. If the core voltage is too low for a certain clock frequency, the transistor gates will not have time to reach the threshold voltage before it switches off again. This can cause the circuit to become unstable. Moreover, if the voltage is too high compared to the clock frequency, unnecessary energy might be wasted due to the quadratic relationship between voltage and power consumption.

The processors used in present-day devices, commonly implement a manufacturer-specific table defining which voltage that is recommended for each frequency. This means that the clock frequency and the core voltage are in most CPUs changed together. Table 2.1 lists the operating voltage for each clock frequency for the ARM Cortex-A15 processor, as implemented in the Linux 4.14 kernel [25].

Table 2.1: DVFS levels for ARM Cortex-A15

Frequency	Voltage
2.0 GHz	1.312 V
1.9 GHz	1.25 V
1.8 GHz	1.2 V
1.7 GHz	1.162 V
1.6 GHz	1.125 V
1.5 GHz	1.087 V
1.4 GHz	1.062 V
1.3 GHz	1.05 V
1.2 GHz	1.05 V
1.1 GHz	1.0 V
1.0 GHz	0.975 V
0.9 GHz	0.95 V
0.8 GHz	0.925 V
0.2-0.7 GHz	0.9 V

The Linux kernel implements DVFS in the form of frequency governors. A governor is a set of policies that controls the frequency of a CPU core or a cluster of CPU cores and how it is adjusted. The Linux kernel has support for multiple types of governors. Some common ones include the *Performance* and *Ondemand* governors. The *Performance* governor runs the core at the highest available frequency while the *Ondemand* governor adjusts the clock frequency based on the current workload.

2.2.2 Thermal frameworks

In order to keep a CPU from overheating, an operating system commonly features a thermal framework that monitors the temperature dissipated inside a processor. The Linux kernel implements a thermal framework that consists of thermal zones, trip points, and cooling devices. The thermal zones are sensors and the devices that are affected by the temperature they measure. The trip points are temperature limits that dictate when a specific cooling action should occur. These actions are carried out by cooling devices that are physical fans or a piece of software that limits the DVFS levels available to the frequency governors. Trip points can also be configured to turn off cores or the entire system if needed.

2.3 Thermal dissipation

The heat that is generated by a CPU must be transferred away to keep the core from overheating. Therefore, a processor is constructed in several layers that spread out and dissipate the heat. Figure 2.1, visualizes the different layers in a typical processor. The die, the piece of silicon that the chip's functional logic is manufactured on, is where all the processor's heat is generated. For the die not to overheat in an instant, a heat spreader is placed on top of it to transfer the heat away from the die. The heat spreader is generally the part that composes the exterior lid on top of a processor chip and is made of a material with high thermal conductivity, such as copper or aluminum [26]. Underneath the die is a substrate layer that provides base support and some form of connectors so the package can be mounted on a PCB. Together, the die, the heat spreader and the substrate comprise a processor package. A multi-core processor, however, can have many dies inside the same package.

On top of the heat spreader, it is common to place some form of a heat sink, either passive or active, with some thermal interface material (TIM) between to increase the thermal coupling between the layers. A heat sink serves as an extra heat spreader, and its purpose is to increase the amount of heat that is dissipated into the ambient environment.

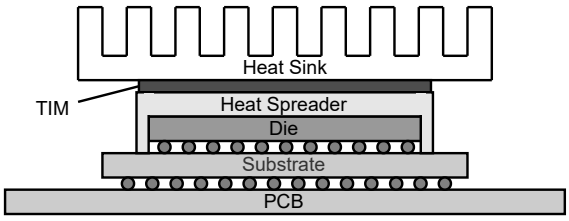

Figure 2.1: Layers in a typical processor setup.

2.3.1 Heat transfer

Heat transfer involves the transport of thermal energy within a medium or between different media due to a difference in temperature. Heat can spontaneously be transferred only from a warmer to a colder medium, as described by the second law of thermodynamics. There are three ways in which heat can be transferred: through conduction, convection and radiation. A processor relies on all three forms of heat transfer to dissipate heat.

Conduction is the transfer of energy within a system or from one object to another due to a temperature difference. Conduction is reliant on materials being

in contact with each other and requires no movement of the matter itself. Fourier's law [27], the law of heat conduction, is described by Equation (2.4). Here, Q is the heat transfer rate (W), k is the thermal conductivity of the material (W/mK), A is the cross-section area in the heat flow direction (m^2) and $\frac{dT}{dx}$ is the temperature gradient (K/m) in the direction of heat flow. Conduction is the main source of heat transfer inside a processor die [28].

$$Q = -kA\frac{dT}{dx} \tag{2.4}$$

Convective heat transfer is the energy transfer across a material boundary by a combination of conduction and a process called advection. Advection is the movement of matter inside a fluid. Therefore, convection relies on fluid matters to function. The following equation dictates the formula for convective heat transfer:

$$Q = hA\frac{dT}{dx} \tag{2.5}$$

In the above equation, Q and A are the same as in Equation (2.4), h is a heat transfer coefficient that depends on many factors, such as the geometry of the object's surface and the fluid's viscosity. Inside a processor, convection does generally not occur, as there are no fluid components. Convection is, however, the main source of heat dissipation between the processor and the ambient environment, usually through the air [29].

Thermal radiation is electromagnetic radiation emitted from surfaces due to their temperature. This form of heat transfer does not require any media to be in direct contact with each other, as heat can radiate through a vacuum. The Stefan-Boltzmann law, shown in Equation (2.6), governs the power transfer from a surface through radiation.

$$Q = \sigma AT^4 \tag{2.6}$$

In Equation (2.6), σ is the Stefan-Boltzmann constant, A is the surface area and T is the surface temperature. It is also common to include an emissivity factor ϵ, as the Stefan-Boltzmann law only holds for black bodies. Emissivity is the relationship between the energy emitted from a surface in the form of electromagnetic radiation and the energy that a black body at the same temperature would have emitted. Aluminum foil and polished metals have a low emissivity in the range of $\epsilon < 0.1$ while, for example, ice and black soot has an emissivity close to 1 [30]. In processors, radiation is not a major contributor to the overall heat dissipation. However, the small amounts of power radiated can still be detected by equipment such as thermal cameras to measure the temperature of a processor.

2.4 Thermal model of a CPU

The heat produced by a CPU is closely coupled to its power dissipation. As shown in Equation (2.4) and (2.5), the amount of heat that is transferred from a processor to the ambient is dependent on the temperature difference between the processor and the ambient. Hence, the heat dissipation can be viewed as a dynamic system that can be represented by the following differential equation:

$$C\frac{dT}{dt} + \frac{T - T_{amb}}{R} = P \tag{2.7}$$

Here, T_{amb} is the ambient temperature and R and C represent the thermal resistance and conductivity of the chip, respectively. Based on this, the thermal-electric analogy can be utilized to model the temperature dynamics of a processor as an RC-circuit. This analogy is throughout thermodynamics and thermal modeling, a common concept to utilize. As described in [31, 32], there exists a similarity between how charge is transferred in electric circuits and how heat diffuses through materials. Therefore, it is not uncommon to represent thermal systems as equivalent electrical circuits. The following relations are commonly modeled:

- Heat is modeled as electrical charge.
- Heat flow is represented by electrical current.
- Temperature difference is represented by voltage.
- Thermal resistance is analog to electrical resistance.
- Thermal conductivity corresponds to electrical capacitance.
- Heat sources are modeled as current sources.

By deploying this analogy, solving problems involving heat transfer becomes easier to breakdown and visualize. Figure 2.2 shows how a dual-core CPU without any fan or heat sink can be represented using this analogy. This model is an abstract and rather simplistic model of a CPU. In reality, there are more factors and components that affect how power is dissipated inside a CPU. However, it serves as a good example of how the thermal dynamics of a processor function.

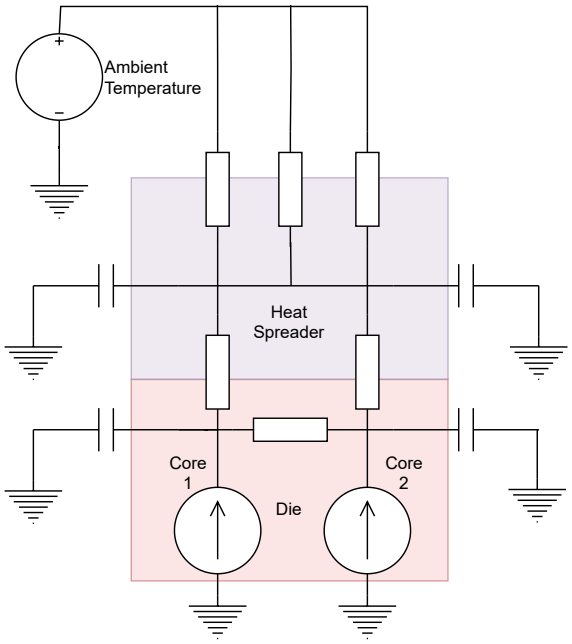

Figure 2.2: An equivalent thermal circuit for a dual-core processor.

In the above example, the power generated by each core inside the die, can both spread horizontally between the cores and vertically to the heat spreader. The capacitors represent how much heat that can be stored inside an object relative to the ambient temperature. In this simplified example, heat can only be dissipated to the ambient through the heat spreader. The three resistors connected between the heat spreader and the ambient are analogous with heat removed through convection, conduction, and radiation, respectively. From a system identification perspective, this type of model is considered a white-box model as it is based on first principles, i.e., the theoretical relationship between the power and thermal dissipation.

Modeling a processor based on first principles and the thermal-electrical analogy requires extensive knowledge about the characteristics of the processor and its environment. For some processors, numerical values for the thermal characteristics of the materials and placement of the processor parts are readily available. However, for many processors, these values are not provided and have to be estimated or measured. In such scenarios, other techniques that do not rely on the thermal characteristics of a processor can, therefore, be considered. In the following chapter, a description of more data-driven gray-box and black-box modeling techniques used in the field of system identification is provided.

3 System identification

The field of system identification deals with modeling dynamical systems using statistical and computational methods. System identification also deals with how to design experiments to capture the dynamics of a system optimally. A common objective for system identification tasks is to reduce a complex physical system to a mathematical or computational model that can be used for prediction and simulation. The purpose of this chapter is to introduce a few concepts related to system identification.

3.1 Basic definitions

To start off, a few basic definitions and concepts should be established. The area of system identification has an extensive terminology that is not always completely consistent. This section will establish a basic description of terminology that is used throughout this thesis [33].

- A system is a conceptualization of a real-world process, such as a physical process or the mechanisms of economics on the stock market.
- A model is in the field of system identification, a relation between measured quantities. Most commonly, how one or more inputs maps to one or more outputs. The relationship is typically expressed as a mathematical formula but can be any type of function, such as a lookup table. A model is, thus, a manageable representation of a system that seeks to approximate the system.
- The difference between a model and the system the model represents is commonly known as the approximation error or prediction error. It is the difference between the output of the actual system and the approximation of the system that the model represents.
- A regressor is an independent variable used as input when estimating a model. A regressor can also be known as a feature, a term that is commonplace in machine learning contexts.

- A regressand is a dependent variable that the regressors are used to predict the outcome of.

- SISO and MIMO are terms that are used to describe the number of inputs (regressors) and outputs (regressands) a system has. SISO stands for Single-Input-Single-Output and MIMO Multiple-Input-Multiple-Output. The terms SIMO and MISO also exist to describe multi-variate systems with either a single input or output.

- Hyperparameters are in system identification and machine learning parameters that are not learned. These are often factors that impact the model structure itself or how it is trained.

- In data-driven modeling and machine learning, training, validation and test sets are commonly utilized. These are data sets that fill a different purpose during the model selection and validation procedures. The training set is the data the model learns from and tries to mimic. The validation set is used to tune the hyperparameters of the model. The test set is utilized to provide an unbiased evaluation of the model's ability to generalize to unseen data. The model's performance is generally assessed using the test set. The test set is sometimes also known as the holdout set.

- Overfitting is when a model is too closely fit to the data it is trained on and fails to generalize to unseen data. The opposite of overfitting is underfitting, and it can happen when a model is too simple and fails to capture the underlying structure or function from the data.

- A system is commonly divided into two parts: the deterministic part and the stochastic part. In a deterministic system, the output is fully determined by the input values and the initial conditions. For a system to be considered stochastic, one or more components of the system have some randomness associated with it. A completely deterministic system will always produce the same output for a given input. On the contrary, an utterly stochastic system always produces an entirely random output given the same input.

3.2 Types of modeling

As describe in Section 1.2, modeling approaches can be categorized into three groups: white-box, gray-box and black-box models. In addition to this classification there are a few more way that models can be categorized. This section will define some common modeling classifications that are used throughout the field of system identification.

3.2.1 Dynamic and static modeling

The distinction between a static and a dynamic system is whether the system incorporates some sort of memory or hysteresis effect. The behavior of a dynamic system is time-dependent and is, therefore, not only determined by the input's current values but also by the input's past values. A dynamic system is commonly represented by a differential equation for continuous-time models or a difference equation for discrete-time models. A static system is only dependent on the current input values and is, therefore, time-invariant.

3.2.2 Online and offline modeling

For dynamic modeling, there is yet another distinction. A model of a multi-variate dynamic system can be obtained through both offline and online methods. These are also referred to by other names, such as parallel and series-parallel, and simulation and prediction. An offline method utilizes past estimates of the outputs to predict the next output. This means that the error is passed on from time step to time step. An online method, however, relies on past measurements of the output to predict the output for the next time step.

3.2.3 Parametric and non-parametric modeling

Modeling approaches can additionally also be viewed as belonging to one of two classes: parametric and non-parametric approaches. The distinction between them is whether the method makes any assumptions about the model's underlying structure. A model structure is an analytical representation of a system usually based on some knowledge about the system under evaluation [34].

For parametric identification methods, a model structure is given and the objective is to numerically optimize a set of parameters to fit the model to a set of data.

Parametric system identification assumes that there is a finite set of parameters, and thus, the model complexity is bounded even if the amount of training data is unbounded. Linear regression and simple neural networks are both examples of parametric modeling approaches. Some advantages of using parametric approaches include: they are often faster to train, more straightforward to understand as well as interpret and they often require less training data. A disadvantage with this type of system identification is the constraint set by the underlying model structure. If the model structure is poorly defined from the start, the parametric approaches will struggle to fit the model to the data. This type of modeling, therefore, requires some knowledge about the dynamics of the system under evaluation.

Non-parametric identification methods are approaches where an underlying model structure or set of model structures are not provided and the method is free to learn any mapping between the input and output data. Non-parametric models rely entirely on training data to make predictions about unseen data. Thus, this type of modeling has a complexity that is bounded by the amount of training data. That is, increasing the amount of data the model is trained on will increase the complexity of the model. k-Nearest Neighbor (KNN) and Support Vector Machines (SVM) are examples of non-parametric methods. Some benefits of utilizing this class of modeling include that little to no knowledge about the system dynamics has to be known a priori and very complex models can be modeled using this approach. However, non-parametric methods usually require large amounts of data to be useful and training time can, therefore, be significant. SVM, for example, has a training time complexity of up to $O(n^3)$, where n is the number of samples [35].

This thesis seeks to investigate modeling approaches to temperature dissipation in processors. The system dynamics of a processor is rather well understood and has been covered by many researchers previously. Therefore, a general idea of the underlying model structure is easily obtained. For this reason, only parametric models are considered in this thesis.

3.2.4 Linear and nonlinear modeling

Another essential classification in system identification is the distinction between linear and nonlinear modeling techniques. Linear models are representations of a system that consists of linear combinations of an input sequence or signal. A system is linear if it has a pair of essential properties: additivity and homogeneity. A system is said to be homogeneous if a change in the amplitude at the input yields an equal

change in the amplitude at the output. A system is considered additive if two or more inputs can be passed through a system separately and their collected sum produces the same result as passing all inputs simultaneously. Linear systems are easy to analyze and synthesize and play an important role in statistical modeling and system identification [13].

Nonlinear systems, the mischievous counterpart of linear systems, are defined by the absence of any of the properties that define a linear system. This makes nonlinear system identification an extensive topic, as every system that is not considered linear is, per definition, nonlinear. The two ensuing sections give a description of linear and nonlinear modeling and introduce a few common system identification techniques for both classes.

3.3 Linear model identification

As aforementioned, the properties of homogeneity and additivity define a linear system. Based on these properties, another important characteristic of linear systems can be outlined. A linear system's output response from a linear combination of inputs is the same as a linear combination of output responses to each individual input. Thus, a linear system can be described only by its impulse response. The impulse function contains all frequencies, therefore, the impulse response contains all information to describe a linear time-invariant (LTI) system [33]. From the impulse response, the transfer function for a system can be extracted by applying the z-transform. In an ideal, completely deterministic world, this would have been sufficient to model any LTI system. However, most systems have stochastic components in the form of noise. This method is, therefore, not always sufficient to describe all systems. Thus, an linear model is instead commonly defined through the input transfer function $G(z)$ and the noise transfer function $H(z)$ and its probability density function $f_e(.)$. This yields a formula for predicting the next output of a discrete-time linear system, as shown in Equation (3.1).

$$y(n) = G(z)u(n) + H(z)\varepsilon(n) \tag{3.1}$$

A common way to represent the transfer functions $G(z)$ and $H(z)$ is to represent them as rational functions on a polynomial form, as shown in Equation (3.2).

$$A(z^{-1})y(n) = \frac{B(z^{-1})}{F(z^{-1})}u(n) + \frac{C(z^{-1})}{D(z^{-1})}\varepsilon(n) \tag{3.2}$$

The two transfer functions are split into a numerator and a denominator part. Additionally, a polynomial A represents the effect previous values of the output has on the next output. Furthermore, z^{-1} denotes the backward shift operator. I.e., it shifts a variable back one time step each time it is applied. This model structure is known as the general-linear model.

3.3.1 General-linear model

The general-linear model is a type of input-output model that specifies the relationship between the input and the output directly. A perhaps more intuitive way of representing such a model is with a box-model layout, as shown in Figure 3.1.

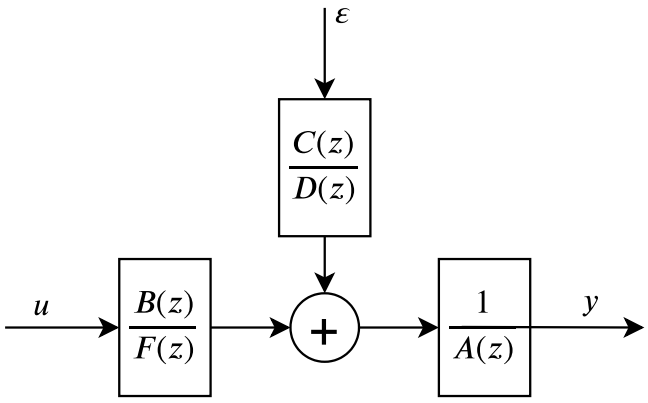

Figure 3.1: Box layout of the general-linear model.

The polynomials A, B, C, D and F each represent how past and current values affect the system. The polynomial A describes how a finite set of past outputs affects the next output of the system. B and C represent how the past and current inputs values and noise respectively affect the output of the system. F and D describe how the past values of the output independently affect the contribution of inputs and noise into the system. Furthermore, this model also assumes that the inputs and the noise are exogenous in regards to other inputs and noise.

This model serves as a theoretical model to describe the dynamics of a general LTI system. In many applications, however, this model is usually too general to be used in practice [33]. Instead, simplified models that utilize a subset of the polynomials in the general-linear model have been devised. Following is a description of some model structures that are commonplace in system identification and optimization:

FIR

Setting the polynomials A, C, D and F to 1 yield the simplest type of linear model, the finite impulse response (FIR) model. When the system output is dependent on only its current input, previous inputs and some noise, this model is akin to using the systems impulse response as the transfer function, as it specifies the system as a linear combination of current and past inputs. Equation (3.3) shows the formula describing this type of model.

$$y(n) = B(z^{-1})u(n) + \varepsilon(n) \tag{3.3}$$

The FIR structure is commonly utilized throughout signal processing and process engineering.

AR

Another model structure is the auto-regressive (AR) model. It is used to model uni-variate systems where a single variable is only dependent on its own previous values. Applied to system identification, this can be utilized to model systems where the output is only dependent on past outputs with some added process noise. To achieve this model, the polynomial are chosen, such that $B = 0$, $C, D, F = 1$. This results in the following equation:

$$y(n) = \frac{1}{A(z^{-1})}\varepsilon(n) \tag{3.4}$$

This model is commonly applied to time-series forecasting, such as stock market predictions.

ARX

An extension of the AR model structure is the autoregressive with exogenous inputs (ARX) model. In this model structure, the output (y) is dependent on past outputs and some noise (ε), just as the AR structure. However, it is also dependent on a set of inputs (u).

$$y(n) = \frac{B(z^{-1})}{A(z^{-1})}u(n) + \frac{1}{A(z^{-1})}\varepsilon(n) \tag{3.5}$$

The ARX model structure assumes that the noise is added inside the process, and thus, has the disadvantage that the noise is modeled as being part of the system dynamics. Therefore, this model is only suitable when the signal-to-noise ratio is high.

ARMAX

As an extension to the ARX structure, a structure that deals with stochastic disturbances that are added early in the process, the autoregressive-moving-average with exogenous inputs (ARMAX) has been devised.

$$y(n) = \frac{B(z^{-1})}{A(z^{-1})}u(n) + \frac{C(z^{-1})}{A(z^{-1})}\varepsilon(n) \tag{3.6}$$

This is more flexible than the ARX model, as it considers disturbances separately from the process dynamics. The ARMAX model is commonly used to model system where disturbances are introduced early in the process, such as the wind's effect on an airplane.

OE

The Output-Error (OE) structure is a model that specifies the system dynamics completely separate from the noise. Equation (3.7) shows that this model structure only depends on two polynomials B and F, with the noise being separate from the model.

$$y(n) = \frac{B(z^{-1})}{F(z^{-1})}u(n) + \varepsilon(n) \tag{3.7}$$

The OE model is mainly utilized when there is a significant stochastic component in a system. It is also suitable for performing long-term predictions of dynamical systems [36].

3.3.2 Identifying input-output models

There are several approaches to solving models based on any of the input-output structures presented above. One common approach, however, is least squares optimization. The least squares method has a linear and a nonlinear variant depending on whether the errors are linear or not for the unknowns. The linear variant can be applied to solve FIR and ARX models, as they model the noise as being part of the system dynamics. The nonlinear variant is usually based on iterative methods. In each iteration, the solution is approximated with a linear solution. Thus, the basic calculations are similar in both cases. For ARMAX models, the errors depend on the current time step. Therefore, they will, in most cases, require nonlinear least squares optimization for a solution to be found [37]. A drawback with the nonlinear least squares method is that it does not always converge to the global minimum,

unlike the linear variant, which is globally concave and hence does not suffer from non-convergence [38].

3.3.3 State-space model

In the previous subsections, a number of model structures were presented. Those belonged, however, all to the input-output family of model structures. This family of model structures can often be cumbersome as the complexity increases as multiple inputs and outputs are introduced. Another common way to model a system is the state-space representation. The state-space model relies on the inputs, outputs and noise model as well as a set of hidden states. These state variables are used to describe a system as a set of difference equations, instead of specifying the relationship between each input and output separately. In Equation (3.8), the discrete-time state-space representation is shown.

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) + K\varepsilon(n) \\ y(n) &= Cx(n) + Du(n) + \varepsilon(n) \end{aligned} \tag{3.8}$$

Here, y, u and ε are the same as in previous model structures. However, the state vector x is introduced. It contains the internal state of the system. The length of x is the order of the system. The matrix A is the system matrix and specifies how the previous state affects the next state. The matrix B is the input matrix and specifies how each input in the input vector u affects the next system state. The matrices C and D represent how the hidden states and the input affect the output of the system, respectively. The matrix K represents the disturbances' effect on the system state.

3.3.4 Identifying state-space models

There are several algorithms that have been developed to identify state-space models. N4SID and MOESP are two that are commonly used throughout the field of system identification [39]. A benefit of using an algorithm such as N4SID or MOESP is that they are guaranteed to converge to the global minimum, non-iterative and numerically stable [40]. Some input-output approaches, such as OE and ARMAX, are iterative approaches that can easily converge to a local minimum [37].

3.4 Nonlinear model identification

In certain scenarios, a linear model might not be sufficient to capture or approximate the dynamics of the system. The nonlinear counterparts to the linear model structures, rely on the assumption that there is a nonlinear relationship between the output of a system and past inputs and outputs of the system as well as the past and present values of a noise component. Since nonlinearity is defined as everything that is not linear, nonlinear modeling is a very extensive class of model structures. For this reason, it is virtually impossible to provide a complete overview of all aspects and possibilities of nonlinear modeling. Therefore, this section will provide only some common distinctions made in nonlinear modeling as well as a few typical model structures and approaches to nonlinear system identification.

3.4.1 Extension to the general-linear model

The linear model structures presented in the previous sections can be extended with a nonlinear function. In Figure 3.2, a nonlinear variant of the general-linear model is shown. This is perhaps an oxymoron. However, this model structure serves as a good example of how a nonlinear model can be devised with a linear model as a basis.

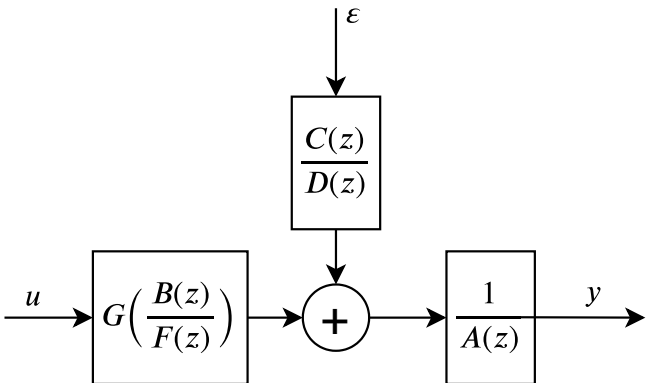

Figure 3.2: Box layout of a nonlinear modification to the general-linear model.

In Figure 3.2, the nonlinear function expansion $G(\cdot)$ maps a set of regressors to a new set of nonlinear regressors. Depending on the configuration of the polynomials A to F, model structures such as NFIR, NARX and NARMAX can be formulated. The function mapping $G(\cdot)$ can be any function, for example, polynomial, piece-wise linear, radial basis function or a wavelet.

3.4.2 Hammerstein and Wiener models

A nonlinear model can also be represented by creating block-oriented structures where each block introduces a specific function. The Hammerstein and Wiener models are two well-known examples of such model structures where a nonlinear block and a linear block are connected in series. A Hammerstein model has a static nonlinear component followed by a dynamic linear component. In the Wiener model, the order is reversed. A combination of the two can be created by adding a static nonlinearity both before and after a linear component [41].

3.4.3 Nonlinear state-space model

Constructing a state-space variation of the aforementioned nonlinear models is also a possibility. A state-space model with an input nonlinearity yields the set of equations shown in Equation (3.9). The function $G(\cdot)$ maps the regressors (u) to a new set of nonlinear regressor (v).

$$\begin{aligned} x(n+1) &= Ax(n) + Bv(n) + K\varepsilon(n) \\ y(n) &= Cx(n) + Dv(n) + \varepsilon(n) \\ v(n) &= G(u(n)) \end{aligned} \tag{3.9}$$

If both the hidden states x and the function mapping $G(\cdot)$ are unknown, this type of model becomes highly complex to identify. As no convergent identification methods such as N4SID exist for this nonlinear case, the simpler input-output models are commonly preferred [42].

3.5 Neural networks

Parametric system identification can also be performed utilizing Artificial Neural Networks (ANN). This is a collective term to describe a number of self-learning algorithms that mimic the function of biological neurons, such as in a human brain. An algorithm that emulates biological neural networks are often deployed to deal with problems that are difficult to solve with conventional mathematical and computational methods. Some examples of applications include image classification, natural language processing and time series forecasting. A neural network must be trained before it can be used. Therefore, model identification utilizing a neural network is

commonly considered a black-box technique in the field of system identification [33, 13, 43].

3.5.1 Feedforward networks

A common form of ANN is the feedforward neural network. It is one of the earliest forms of ANNs that were developed [44]. In this network structure, the neurons are all connected in the same direction without any loops or feedback. The neurons are commonly arranged into layers. When a feedforward network only contains a single layer it is known as a single-layer perceptron. Networks consisting of several layers are known as multi-layer perceptrons.

A feedforward neural network has three types of layers: input layers, hidden layers and output layers. The input layer takes the input data for the neural network. The hidden layers are located in the middle of the network and perform operations to transfer the data from the input to the output of the network. The output layer transfers the data from the hidden layer to the outputs of the network. Figure 3.3 shows an example of a multi-layer perceptron with a single hidden layer. The network maps an input of size 3 to an output of size 2.

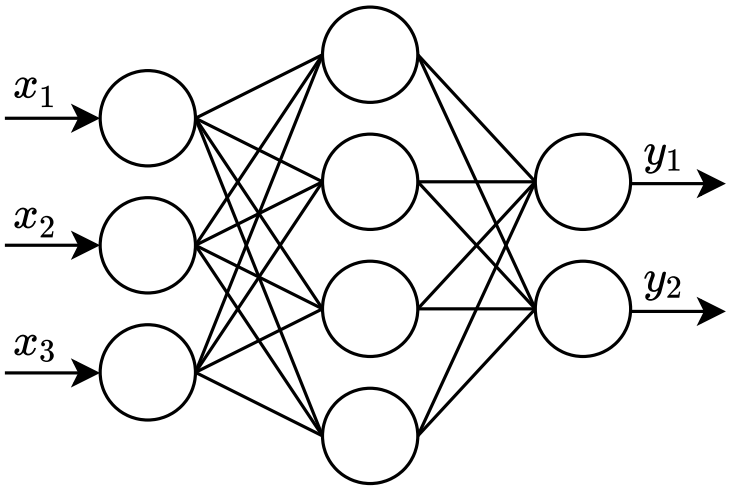

Figure 3.3: A multi-layer perceptron with one hidden layer.

A neural network commonly only consists of a single input and output layer but can comprise many hidden layers. Networks that utilize more than one hidden layer are commonly known as deep networks. The individual components of the layers are referred to as nodes or neurons [45]. The structure of a neuron is visualized in Figure 3.4.

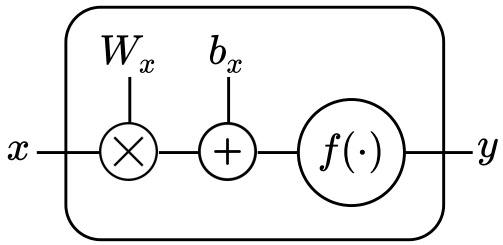

Figure 3.4: The structure of a neuron.

The neuron takes an input x that contains the values from the previous layer or the input values of the network. The matrix W_x is multiplied with x and contains the weights for each input element in x. The size of W_x is determined by the size of the input vector x and the output vector y. After the multiplication, a bias vector b_x is added and an activation function $f(\cdot)$ is applied on the resulting vector to produce the vector y. This value is then passed to the next layer or to the output of the network.

In a linear neuron, the activation function $f(\cdot)$ is omitted. However, a linear neural network can only learn linear relationships and will require large networks to approximate a nonlinear function. To learn nonlinear relationships, the activation function $f(\cdot)$ can be any function, but is usually a continuously or piece-wise differentiable function, such as a logistic function or a rectifier [45].

Neural networks are trained by adjusting the weights and biases of the neurons, commonly through an optimization algorithm called gradient descent or a variation thereof. With standard gradient descent, training is performed by calculating how large the error is and what the gradient of the error is. The error is calculated using a loss function, which is a differentiable function that quantifies the error. Mean Squared Error (MSE) and Mean Absolute Error (MAE) are two frequently applied loss functions. When MSE is chosen as the loss function, the optimization is essentially the same as least squares optimization.

During training, the weights and biases of each neuron are adjusted in a direction that minimizes the error. This procedure is iterated until the algorithm converges to a solution or until it is manually stopped. There is a myriad of algorithms for training various types of neural networks. One such algorithm is stochastic gradient descent. Instead of calculating the actual gradient, it calculates an approximated gradient from a subset of the data. Some optimizers also utilize the second derivative of the loss function. Levenburg-Marquardt is a popular example of such an algorithm. Methods that utilize the second derivative are, however, usually not preferred for neural networks as they have high space and time complexity [46].

3.5.2 Recurrent networks

A Recurrent Neural Network (RNN) is a type of neural network that integrates feedback loops. This allows the neural network to capture the dynamic components of a system. RNNs are better at dealing with sequential data as it incorporates memory variables that keep information between time step [45]. Figure 3.5 shows a simple RNN neuron that takes both an input x and a state variable h from the previous time step.

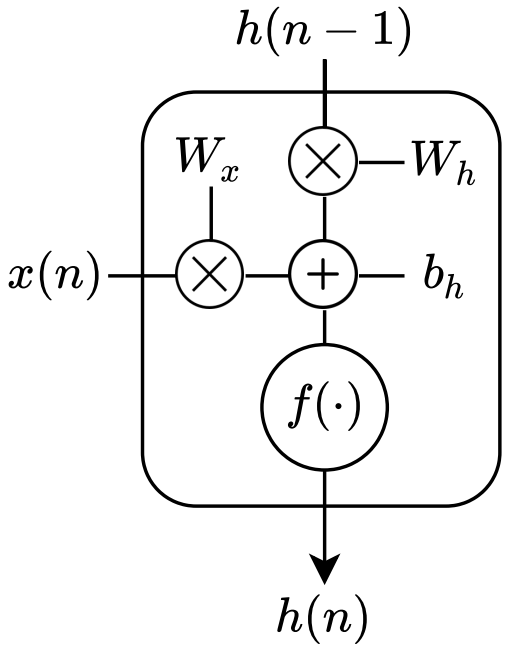

Figure 3.5: A simple recurrent neuron.

A problem with this type of simple recurrent structure is that it is sensitive to the vanishing gradient problem. A vanishing gradient can occur when several layers with activation functions that feature regions with small gradients are added to a neural network. Some activation functions, such as the sigmoid, take large input values and maps them to values between 0 and 1. Thus, even if there is a large change in the input, it might only trigger a very small change in the output. Therefore, the loss function gradient can get vanishingly small, causing difficulties when training the network [45].

More complex recurrent node structures have been suggested to combat the vanishing gradient problem. Two frequently utilized structures are the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) [47]. The LSTM was first suggested in 1997 [48]. It features two outputs, an output state just as

the simple RNN structure, but also a cell state. The cell state is essentially the long-term memory of the LSTM. The inner workings of an LSTM neuron are more complex, as shown in Equation (3.10).

$$\begin{aligned} i(n) &= \sigma(W_{ix}x(n) + W_{ih}h(n-1) + b_i) \\ f(n) &= \sigma(W_{fx}x(n) + W_{fh}h(n-1) + b_f) \\ \tilde{c}(t) &= \tanh(W_{cx}x(n) + W_{ch}h(n-1) + b_c) \\ o(n) &= \sigma(W_{ox}x(n) + W_{oh}h(n-1) + b_o) \\ c(n) &= f(n)c(n-1) + i(n)\tilde{c}(n) \\ h(n) &= o(n)\tanh(c(n)) \end{aligned} \tag{3.10}$$

A forget variable f that controls which information to forget between time steps is introduced. The variables i and $\tilde{c}$ control which and how much information that should be saved in the cell state variable c. The variable o controls what information in c that is sent to the output state h.

The GRU was introduced by Cho et al. in 2014 [49]. It has a simpler structure and fewer parameters than the LSTM. Furthermore, it does not feature a cell state. Equation (3.11) shows how a gated recurrent unit is structured.

$$\begin{aligned} z(n) &= \sigma(W_{zx}x(n) + W_{zh}h(n-1) + b_z) \\ r(n) &= \sigma(W_{rx}x(n) + W_{rh}h(n-1) + b_r) \\ \tilde{h}(n) &= \tanh(W_{hx}x(n) + W_{hr}r(n)h(n-1) + b_h) \\ h(n) &= z(n)h(n-1) + (1 - z(n))\tilde{h}(n) \end{aligned} \tag{3.11}$$

The variable r is a reset variable that controls how the information from the previous time step is weighted against the new information in x. The variable $\tilde{h}$ represents new information that is learned from x and the previous state h. Finally, z is an update variable that regulates how much information from the previous output state h and $\tilde{h}$ that is sent to the next output state h.

Although the LSTM has a higher complexity, the performance of GRU-based models are comparable to utilizing LSTM for some applications such as speech processing and polyphonic music modeling [50].

4 Related Work

This chapter presents a review of previous research that has been performed in the fields of nonlinear system identification and thermal modeling of processors. The purpose of this chapter is to gather an understanding of which approaches to system modeling and identification that are commonly utilized and how they are applied to model the dynamics of a processor. In the end, an assessment of the various approaches will be presented and a few candidates will be selected for implementation in a comparative study.

4.1 Thermal modeling

The field of thermal modeling has seen a large number of approaches. Looking through the literature, methods that utilize everything from theoretical white-box models to black-box models using neural networks have been proposed. The following are examples of such modeling techniques.

There have been a few articles that propose white-box approaches that are based on the theoretical equation that govern the power and heat dissipation of a processor, such as [51, 52, 53]. These implement a bottom-up technique, where the thermal model is based on the layout of the SoC, the conditions of the external environment and the conductive properties of materials. These approaches simulate the thermal dissipation directly at chip-level, however, with some level of abstraction. This type of modeling relies heavily on the accuracy of the technical parameters and how much detail is lost through abstractions and simplifications.

Another common approach that has been utilized in some articles [54, 55, 56, 57] is to model a chip using the thermal-electrical analogy. The chip is broken down into small parts, where each part is represented as a combination of current sources, resistors and capacitors. A common tool for this is HotSpot developed by Huang et al. [58]. These approaches also rely heavily on knowledge about the characteristics as well as the location of components within the chip.

The previously presented approaches have been towards the white-box end of the modeling spectrum. An example of a more data-driven learning approach is

the Generalized-Pencil-of-Function (GPOF) method. By utilizing this method, the transfer function of the system can be extracted from the step response. In [59, 60], Li et al. apply the GPOF method to model the temperature of a multi-core processor. A logarithmic sampling scheme is implemented to provide a more precise representation of the fast changes in temperature.

A few articles have also been published where researchers apply gray-box identification approaches to model the thermal characteristics of a processor. Beneventi et al. [31] propose an approach where a multi-core processor is modeled as a thermal-electrical circuit. In their approach, the processor is divided into blocks that correspond to each core and the section of the copper heat spreader directly above each core. The parameters of the model are then optimized using an Output-Error approach. A similar approach was proposed by Aguia et al. [61]. They suggested an implementation where the cores of a multi-core processor and the cache memory are represented as blocks in a thermal-electrical-equivalent circuit. The subspace identification method, N4SID, is then applied to find the optimal parameters for the model. Another approach that utilizes a state-space identification method has been proposed in [62]. Here, the researchers deploy a piece-wise linear subspace identification method that estimates a linear model for each temperature range. Shetu et al. [63], however, suggest a different approach with a polynomial model for approximating the temperature of a CPU. In their study, a thermal model is constructed by creating polynomials based on the size and intensity of the workload.

Approaches towards the black-box end of the system identification spectrum that utilize neural networks have also been proposed. Vincenzi et al. [64] and Sridhar et al. [65] have suggested two similar implementations where the thermal dynamics of an integrated circuit is predicted using ARX linear neural networks. These approaches were shown to be effective at simulating heat flow in 3D-dimensional and highly granular, integrated circuits. An approach to simulating the heat dissipation in processors using a feedforward neural network has also been proposed in [66]. The researchers compared the performance of a Gaussian process model, a NN model and a linear regression model. The results showed that the neural network model outperformed the linear model in terms of prediction accuracy by 30%, but was approximately three times as computationally expensive. The Gaussian process model also showed good prediction accuracy. However, it had twice as much computational overhead as the NN model. Another interesting approach was tested by Pérez et al. [67] in an article from 2018. They compared recurrent and feedforward neural network structures for thermal prediction of immersive cooling computer systems.

A simple feedforward ANN structure was compared with two other structures utilizing LSTM and GRU layers in an FIR configuration. The temperature predictions in this study were based on the core frequency and processor utilization measurements from the past minute. The results revealed that the shallow GRU and LSTM structures produced the lowest prediction error.

4.2 Power modeling

Many of the approaches to thermal modeling presented in the previous section rely on power measurements to predict the temperature of a processor. There are examples in the literature of research that proposes methods for predicting the power dissipation of a processor. In [68, 69, 70], Walker et al. utilize core frequencies, core voltages and event counters to train a linear regression model to predict the power consumption of a multi-core processor. The events used were, for example, cycle counter, bus and cache accesses. Zhang et al. [71] implement a similar approach. They also build a linear regression model based on data collected from a CPU. Unlike, Walker et al., however, they utilize the idle states and idle time of each core. Another similar approach has been suggested in [72] by Balsini et al. The Italian researchers deploy a genetic algorithm to find the optimal parameters for a function that represents the theoretical relationship between power dissipation and quantities such as the core voltage and clock frequency.

A neural network-based approach to modeling the power of CPU has been proposed in [73]. In this paper, Djedidi et al. construct a power model for an ARM-based mobile device using a NARX structure. The model is constructed based on data such as core frequency, screen activity and network usage.

4.3 Conclusion

The approaches presented in this chapter reveal that a wide range of methods has been applied to model or simulate temperature development in processors, computing systems and adjacent areas. Most approaches have relied on more classical gray-box system identification techniques. However, articles published in recent years have focused more on machine learning techniques and especially neural network approaches. Given the amount of focus that is given to machine learning and artificial intelligence in the research of today, it will be interesting to see what lies ahead for the field of thermal modeling and nonlinear system identification.

In this thesis, the first research question aims to identify which approaches that would be suitable for thermal modeling of heterogeneous processors without relying on measurements of the power consumption. The papers presented in this chapter suggest a few system identification methods that can be applied in this thesis project. Most of the white-box and gray-box approaches utilize power as an input variable or regressor. Since the thermal properties and the floorplan are not directly available for the ARM CPUs, a white-box approach is not suitable for implementation in this thesis. Many gray-box approaches also relied on the close-to-linear relationship between temperature and power. This also makes these approaches less appealing when the objective of this thesis is to perform modeling based only on frequency and processor utilization. However, there were some examples of articles presented in this chapter that relied on the theoretical relationship between frequency, voltage and utilization to estimate the power dissipation of a processor, such as the articles published by Walker et al. [70] and Balsini et al. [72]. Combining such an approach, with a linear model identification technique, such as the N4SID method suggested in [61], could be a suitable approach.

Other approaches that have produced promising results are neural network-based approaches. The neural network approaches proposed in [66, 65], for example, are approaches that would be interesting to combine in this project. A neural network in an ARX structure, could through the addition of a nonlinear hidden layer, learn to replicate the nonlinear dynamics of the heterogeneous processor. This would create a Hammerstein type of NARX model. Furthermore, as the dynamics of a heterogeneous processor is rather deterministic and the noise component in measurements can be expected to be rather low, an ARX-based model should be suitable.

Recurrent neural network approaches have not seen much attention in applications related to thermal modeling of computing systems. However, there is one paper published by Pérez et al. [67] that demonstrates an interesting approach where an RNN model is trained in an FIR structure. This could also be a very suitable option for the system identification task in this project.

Based on this, three model structures have been selected. The first is a nonlinear state-space model structure using nonlinear regressors. The second is an NARX approach, where a neural network is recursively trained to predict the temperature. The third approach is an FIR model structure that utilizes an RNN layer to predict the thermal dissipation.

5 Materials and methods

This chapter covers how the experimental environment was set up and how the study was conducted. The hardware platform and workload application are presented. Moreover, the data collection procedures, model selection and validation strategies are also described.

5.1 Experiment setup

For this study, a desktop experiment setup for bench-marking and measuring the temperature of a heterogeneous processor was created. Following is a description of all parts in the experimental setup. Figure 5.1 shows the entire setup that was used to generate and gather data in this study.

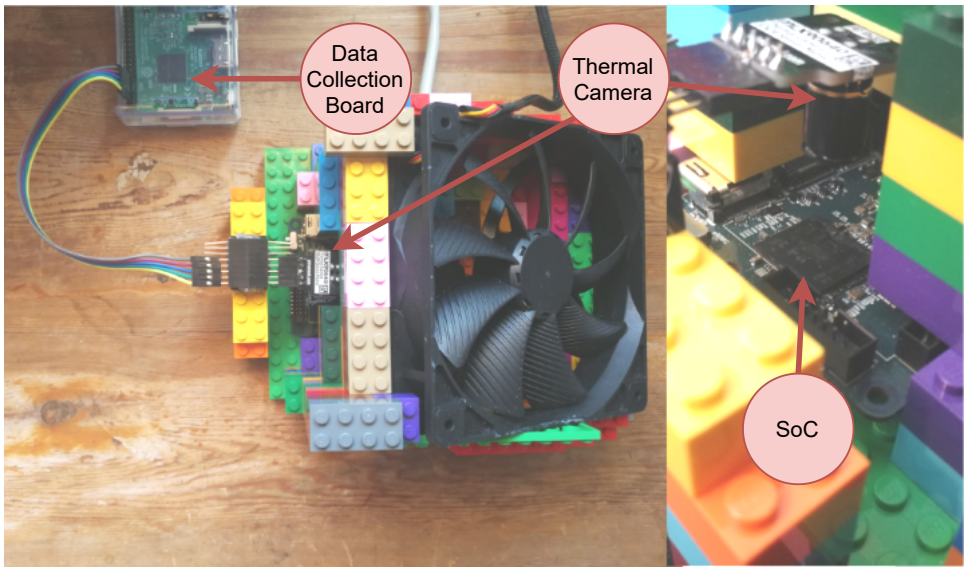

Figure 5.1: The experimental setup.

The setup features four parts: the heterogeneous SoC that is the system under test, a thermal sensor, a cooling fan, and a data collection unit. The remainder of this section will detail the parts of this setup and how data was collected from the heterogeneous processor.

5.1.1 Hardware platform

The Odroid-XU4 was selected as the heterogeneous processor under test in this thesis project. It is a single-board computer the size of a credit card that features an Exynos 5422 SoC manufactured by Samsung. It has eight cores configured in two clusters and runs the Ubuntu operating system based on the Linux 4.14 kernel. The Exynos 5422 implements the big.LITTLE heterogeneous computing architecture developed by ARM. In this architecture, one cluster is more potent in terms of computing power, but also thirstier in terms of power dissipation. The other cluster is smaller and has less computing power while being more energy-efficient. In this heterogeneous processor, an ARM Cortex-A15 is implemented as the big cluster and an ARM Cortex-A7 is implemented as the little cluster. The block diagram of the Exynos processor is shown in Figure 5.2. The SoC also has 2 GB of DDR3 memory as well as a Mali-T628 GPU. However, the GPU is not utilized or considered in this thesis.

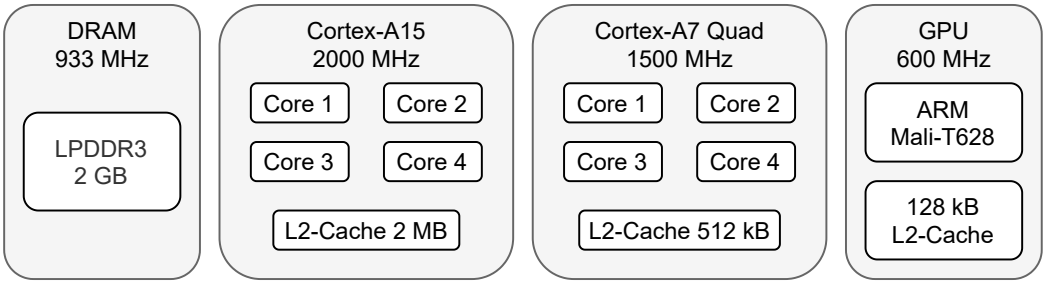

Figure 5.2: Internal block diagram of the Exynos 5422 SoC.

This single-board computer allows for control of the frequency on a per cluster basis between 200 MHz and 2000 MHz for the big cluster and 200 MHz and 1500 MHz for this little cluster. The operating frequency cannot be controlled independently for each core inside the clusters. The voltage levels can also be set for each cluster. However, in the Linux operating system for this platform, these are set to static values for each operating frequency by the kernel. The operating voltage levels are, therefore, not considered as a variable in the implementations in this thesis.

The Odroid board has been configured to trigger a thermal throttle when the core temperature for the big cores reaches 90°C. This means that the processor's frequency governors will reduce the maximum available frequency when the temperature is reached to prevent the processor from overheating.

5.1.2 Experimental workload

The workload application utilized in this project is an RGB-YUV image conversion. This image conversion was chosen as the workload because it is a highly parallel workload that can be distributed to many cores. The workload implementation was taken from the *stress-ng* bench-marking program [74]. This program, however, could not be utilized on its own, as it did not feature the ability to control the utilization on a per core basis. Furthermore, the core frequencies could not be adjusted by this stress testing suite. Thus, a custom-built stress application has been implemented in this thesis.

The stress application is written in C. It implements the previously described workload. The application takes the utilization of each core and the frequency of each cluster as well as the amount of time it should be executing as arguments. Inside the application, a thread for each core in the system is created. Each core thread runs its assigned workload independently from the other cores.

The software manages the utilization control separately for each core's thread. This is achieved by allotting periods of 10 milliseconds. For each period, work is performed for the specified core utilization. The core thread is then put to sleep for the remainder of the period using the *select* system call. The *select* call allows the processor to sleep without deallocating any of its resources. This method of utilization control requires that the core thread has 100% of the core context and can be sensitive to external processes affecting the utilization. It is, therefore, crucial that all unnecessary background applications and system functions are turned off. This utilization technique was compared with the output of the *htop* utilization monitor and it revealed that the utilization was accurate to within $\pm 1\%$.

The cluster frequencies are controlled using the *Performance* frequency governor. The C application does not adjust the frequencies directly, it sets the maximum allowed frequency and the frequency governor then adjusts the frequency accordingly.

5.1.3 Thermal measurements

Due to the absence of a core temperature sensor for each core on the Odroid-XU4, a different temperature collection scheme had to be devised. As mentioned in Section 2.3, the temperature can be quantified by measuring the emitted energy in the form of infrared radiation. Thermal imaging cameras are specialized sensors that measure the intensity of heat radiated by objects. For the setup in this thesis, the small thermal camera MLX90640 from Melexis has been used. It has a resolution

of only 32 by 24 pixels. Therefore, the camera has been mounted close to the SoC of the Odroid-XU4, as can be seen in the right picture in Figure 5.1. This has been done in order to obtain a more accurate reading of the temperature across the surface or the SoC. The MLX90640 sensor has a temperature range of 40°C to 300°C and an accuracy of approximately ±1°C. Figure 5.3 shows a heat map of the Odroid board when the processor is running at 100% utilization for all cores and the cluster frequencies are both set to 1500 MHz. The region that dissipates the most heat is the region where the big cluster is located.

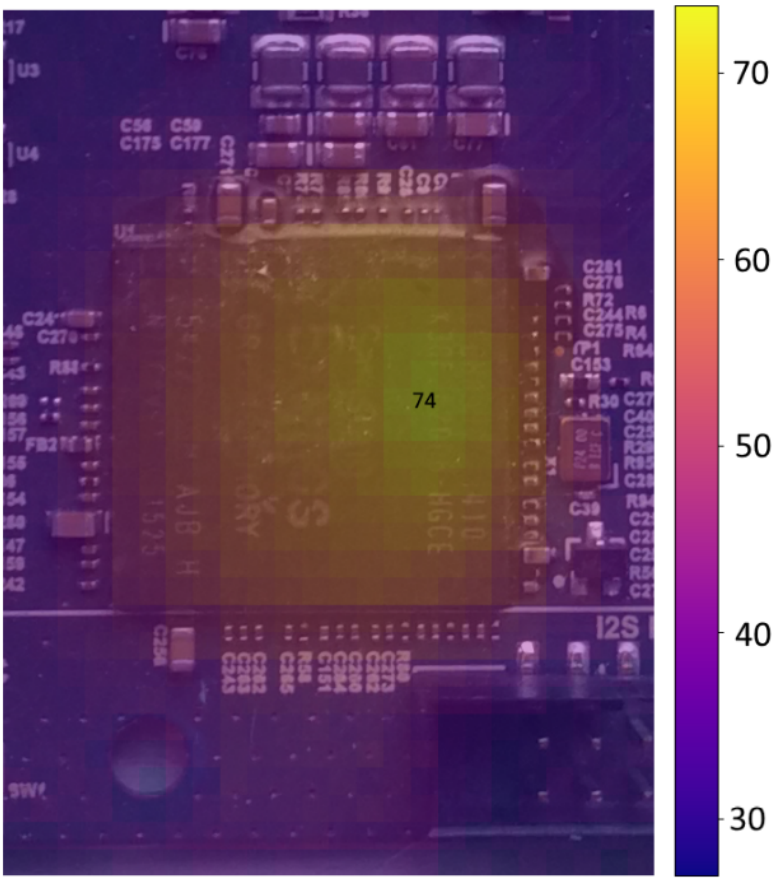

Figure 5.3: Heat map of the Odroid board.

The measurements collected from thermal cameras are dependent on the emissivity of the objects that are being measured. In this experiment, the primary source of heat is expected to be the SoC chip. The emissivity value for a processor IC has been shown to be approximately 0.95 [75]. Because the SoC is the component expected to generate most of the heat on the board, this value is selected for the entire thermal image in this implementation.

5.1.4 Cooling

A drawback of using a thermal camera is that no heat sink can be mounted on top of the SoC. Thus, some external form of cooling had to be implemented in order for the processor not to overheat when running at higher clock speeds. The fan and Lego structure that can be seen in Figure 5.1 provides sufficient directed cooling for the big cluster to be able to run at up to 1900 MHz. In this implementation, the fan is constantly running at 100% speed.

5.1.5 Data collection

A Raspberry Pi has been deployed as the control and data collection unit. It controls the experimental workloads and captures the thermal response. The data from the temperature sensor and the Odroid board were sampled 32 times per second. This sample rate was selected since it is the maximum sample rate for the thermal sensor.

As mentioned earlier, the parameters that can be configured on the board are the frequencies of the two clusters and the utilization threshold for each core. Moreover, one temperature measurement is taken. Therefore, the data collected consists of 11 separate variables; a single regressand and 10 regressors. The regressand is the maximum temperature measured by the thermal camera in degrees Celsius. The thermal image from the thermal sensor is captured at 32 Hz.

The first two regressors collected are the cluster frequency for each of two CPU clusters on the Odroid-XU4. Their values can range between 200 MHz and 1500 MHz for the little cluster and between 200 MHz and 2000 MHz for the big cluster. For the implementation in this thesis, the values have been limited between 1000 MHz and 1900 MHz. The lower limit was included to restrict the number of configurations that will produce a very low thermal output. Including the lower frequencies would have made the data set rather imbalanced in favor of lower temperatures. The higher limit on the big cluster's operating frequency is imposed to assure that no core temperature reaches the thermal throttles point of 90°C. The other eight regressors are the utilization for each core, with a granularity of 25%. This allowed the utilization for each core to be selected at five discrete levels; 0, 25, 50 ,75 and 100%. All regressors were collected at 32 Hz to keep the sampling rate uniform across all collected variables.

The workload is in this thesis constant. Thus, the total number of possible configurations can be calculated using Equation (5.1), where U is the number of utilization levels, C is the number of cores, f_b is the clock frequency of the big

cluster and f_l is the frequency of the little cluster.

$$N_c = U^C f_b f_l \tag{5.1}$$

Each core has five different utilization levels, and the big and little clusters have ten and six discrete clock frequency levels, respectively. For the implementation in this thesis, this yields a total of approximately 23 million possible configurations of the heterogeneous processor.

5.2 Methodology

One of the goals of this thesis was to investigate if model-driven approaches can outperform more data-driven approaches. Therefore, three modeling approaches were selected for a comparative study based on the conclusions in Chapter 4. The comparative study assessed the performance of the three modeling approaches. In many real-world system identification tasks, there is commonly a set of requirements that a model needs to fulfill to be accepted. In this comparative study, however, the objective was to merely find the optimal hyperparameters for each model structure and compare the maximum performance achieved by each modeling approach.

The first modeling approach evaluated in this thesis was a Polynomial N4SID approach, where new polynomial regressors are created based on the 10 original regressors. These new regressors were used to identify a state-space model using the N4SID algorithm. This modeling approach is the most model-driven of the three and is also the most towards the white-box end of the modeling spectrum.

The second approach was based on a Hammerstein-NARX model structure constructed as a neural network. This model has a static nonlinear layer followed by a linear layer with feedback. This modeling approach is also model-driven, however, to a lesser degree than the Polynomial N4SID approach. This is mainly due to the nonlinear part that gives the model more freedom in how it fits to the training data.

The third modeling approach was an RNN-based neural network that is configured in an FIR model structure. This model does not rely on feedback as the other two models. Instead, it predicts the temperature strictly based on the current and past values of the regressors. When utilizing LSTM or GRU nodes, this model structure has to possibility to become very complex and can, therefore, be considered as the most towards to black-box end of the spectrum of the three modeling approaches. It is also the most data-driven model as the nonlinear model structure

gives the model larger freedom to approximate many types of functions.

The performance of these three modeling approaches has been assessed for two different lengths of training data: 1 hour and 6 hours. This section will describe how the data set was created and how the models were selected and validated. The error of each model has been measured using Mean Squared Error (MSE) as the metric. A flowchart of the entire model identification methodology is shown in Figure 5.4. The following subsections will explain in more detail how the different parts visualized in the flowchart were carried out.

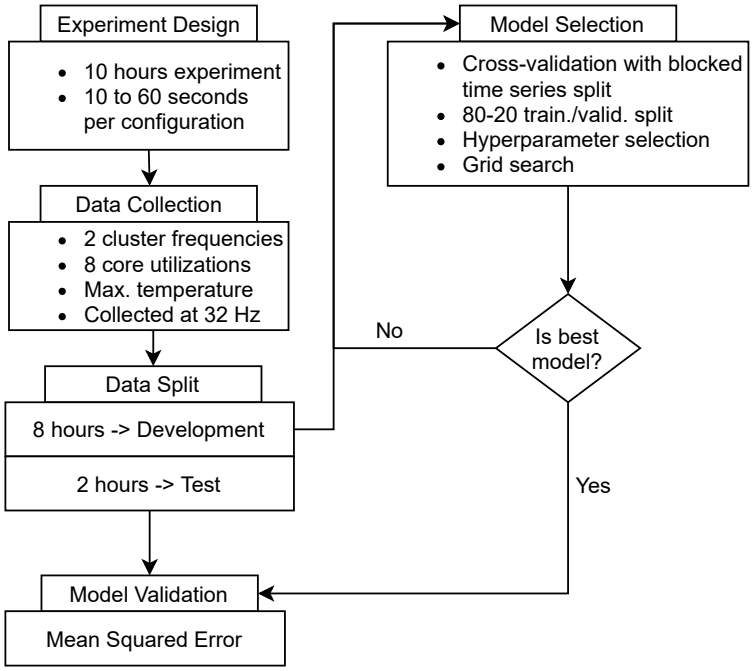

Figure 5.4: Flowchart of system identification procedure.

5.2.1 Experiment design

As a basis for the model selection, 10 hours of data was collected from the Odroid computer board and thermal camera. The data set was created by executing a sequence of randomly selected configurations of the Odroid board using the stress application mentioned in Section 5.1. The configuration was changed after a random amount of time in the range of 10 to 60 seconds. Both the selection of configuration parameters (cluster frequencies and core utilization) and execution period followed a uniform distribution. Throughout the experiment, the ambient temperature was kept steady at 21°C.

The 10-hour-long data set was divided into two sets, a development set and a test set. The first 79% of the data became the development set. This is the portion

of the data that the models were trained on and the models' hyperparameters were evaluated with. The last 20% of the data were chosen as the test set. This is the data set that the final prediction error was assessed upon and was not utilized for model training and selection. A small set of data corresponding to 1% of the total data, lodged between the development and test sets, is omitted to ensure that there is no interference between the development set and the test set. Furthermore, the same data split was utilized for all the compared modeling approaches.

For the model selection and validation procedures, cross-validation was utilized on the development set. Cross-validation is an umbrella term that refers to methods for analyzing how well a model will generalize to a set of independent data. A common practice in machine learning and data science is to utilize k-fold cross-validation. The variable k refers to how many subsets the data is divided into. In this type of cross-validation, the model is trained k times, each time using a different subset as the validation set and the rest as the training set.

In the implementation in this thesis, since the data is a time series, each data point is dependent on the previous data points and the models are assessed for a specific length of training data, a special type of folding procedure called blocked time series split is utilized. Instead of dividing the data set into k equal parts, like in regular k-fold cross-validation, the data is arranged into blocks of a specific length. Figure 5.5 shows a 5-fold version of regular k-fold cross-validation, to the right, and blocked time series cross-validation, to the left.

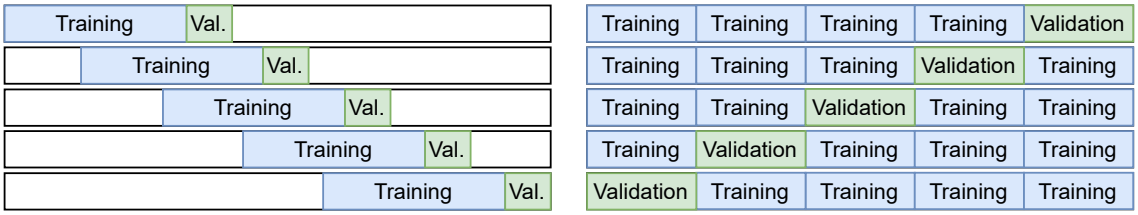

Figure 5.5: Comparison of k-fold and blocked time series cross-validation

In this implementation, the blocks were 1 and 6 hours long depending on which length of training data that was assessed. Just as with regular k-fold cross-validation, some folds can have overlapping training data. However, the crucial thing is that no validation data is shared between folds. Therefore, an experiment length of 10 hours was chosen as it is the least amount of data that allows for dividing the data into multiple folds when evaluating the models' performance on 6 hours of data, while also having a small gap between the test set and any of the training or validation data.

5.2.2 Model selection

The model selection was performed separately for each model structure and training data length. To assess the models' 1-hour performance, 10-fold cross-validation was performed. For 6 hours of data, 4-fold cross-validation was performed. Since the difference in distribution between the validation and training set can be expected to be higher, the shorter the sequences are, more folds were utilized when assessing the 1-hour performance.

The optimal hyperparameters for each model have mainly been selected using grid search. Grid search is an exhaustive search method for testing which hyperparameter combinations that yield the best results for a model. Other search methods also exist, such as randomized search, which tests only a random subset of the hyperparameter combinations.

The same general model selection strategy was deployed for all three modeling approaches. However, for some hyperparameters, additional selection techniques were deployed. Such as the polynomials selected for the N4SID approach and the number of time steps considered for the FIR-RNN approach. The hyperparameter selection process for each modeling approach is detailed in the next chapter.

For the cross-validation, a blocked version of a time series data split was utilized for both data lengths. In Figure 5.6, the cross-validation procedure deployed when assessing the models' 1-hour performance is shown. On the development set, 10 blocks of 1 hour each with equal overlap, were selected. For each block in the cross-validation, the first 80% of each block was utilized as the training set and the remaining 20% as the validation set. For the two neural network-based approaches, the validation set was also utilized to determine when to stop training the models. This is a procedure called early stopping and it is a regularization technique commonly applied to avoid overfitting to the training data.

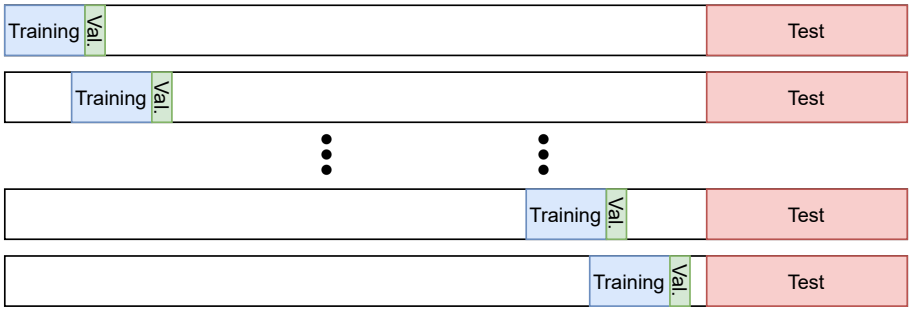

Figure 5.6: 1-hour cross-validation procedure.

When assessing the three modeling approaches performance on 6 hours of training data, a slightly different method was utilized. In the 4-fold cross-validation, two blocks were created in the same manner as for 1 hour of data. The other two blocks were created by reversing the order of the validation and training data inside the block, as seen in Figure 5.7. The validation set comprises the first 20% and the training set the last 80%. This reversed blocked time series split ensured that maximum diversity in the training and validation data is achieved.

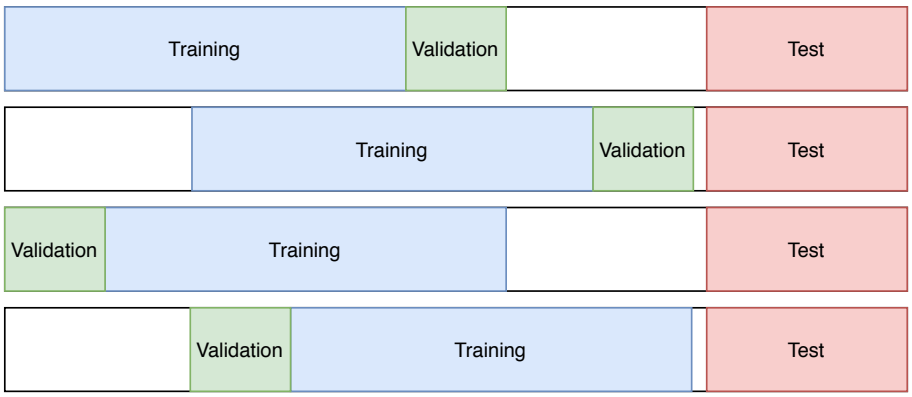

Figure 5.7: 6-hour cross-validation procedure.

Grid search was deployed together with the cross-validation procedure described above to find the optimal values for some of the hyperparameters in each of the modeling approaches. The same procedure was applied to all three modeling approaches. The models were trained using the training set and the error was measured on the validation set. The hyperparameters that yielded the lowest error on the validation set on average across all the folds were selected for the final models.

5.2.3 Model validation

The same cross-validation procedure was deployed to measure the final prediction error of the models. The critical difference is that the N4SID model was trained using the entire block for each fold. Since it does not rely on early stopping on the validation set it can be trained on the validation set as well. Early stopping is, however, utilized for both the other models. Therefore, the number of epochs that the iterative approaches are trained for is not considered as a hyperparameter. Furthermore, these two modeling approaches utilized the same 80-20 divide as during the model selection phase. The final prediction error was evaluated on the test set and the final performance metric was measured as the average MSE across the folds.

6 Implementation

This chapter will provide an explanation of how the selected modeling techniques were implemented and trained. It will also provide additional motivation on why the modeling approaches were selected as well as present how the hyperparameters have been chosen.

In Chapter 4, the review of the state-of-the-art revealed that a multitude of approaches had been proposed for the task of identifying thermal dissipation in processors. To address RQ2, three modeling approaches were, therefore, selected: a state-space identification method, a neural network-based identification method configured in a nonlinear ARX structure and a recurrent neural network approach configured in an FIR model structure. The three approaches are all parametric approaches, but they have differing levels of complexity. The state-space identification method is the most towards the white-box end of the modeling spectrum, as the state-space representation models a system in a way that is similar to a first principle model. The second approach, with a nonlinear neural network, is a black-box approach and has the potential to be very complex. The complexity depends on the number of layers and neurons, as well as which activation functions are implemented. The NARX structure that is implemented in this thesis is, however, rather limited since it will only consist of a single hidden layer. The third approach is also a neural network approach but in a FIR structure based on recurrent neurons. RNNs utilizing LSTM or GRU are complex structures with many internal parameters that grow exponentially with the number of nodes in each layer [45]. Additionally, to capture complex feedback systems, FIR models will generally need to be of a higher order than an ARX or IIR model [76]. Therefore, the FIR-RNN modeling approach is the most complex of the approaches implemented in this thesis.

6.1 Polynomial N4SID

The first model structure is a parametric approach based on the state-space identification method N4SID to estimate a linear state-space model. In Chapter 2, it was described that there is a direct relationship between the power dissipation of a pro-

cessor and its thermal dissipation. This relationship could, therefore, be exploited to construct a linear model of the system. This type of approach has been suggested in both [31] and [61]. In this thesis project, however, the objective is to compare modeling approaches that can predict the temperature based on the clock frequency and the utilization percentage of each core. The power consumption was in Chapter 2 shown to have a nonlinear relationship with the core frequency, the core voltage and the core utilization. While the dynamic power dissipation is linearly dependent on the core utilization, the core utilization cannot, on its own, be used to describe it, as it is also dependent on the core frequency and voltage. Therefore, a nonlinearity had to be introduced to approximate the power dissipation. This nonlinearity was introduced in the form of new nonlinear regressors as polynomial combinations of the core frequency and core utilization. A linear state-space MISO model was then identified by the N4SID algorithm utilizing the nonlinear regressors as the input of the system.

The nonlinear input function for this model was derived from the relationship between the core frequency and the power consumption described in Chapter 2. It was highlighted that the dynamic power consumption is proportional to fV^2. However, the voltage is not a quantity that is considered as it is directly affected by the core frequency. Therefore, the voltage part of the equation can be expressed in terms of its proportionality to the frequency. The voltage levels for the cores in the big cluster are shown in Table 2.1. By using power regression, the approximate relationship was calculated to be $V \propto\!\!\!\sim \sqrt{f}$. Thus, the dynamic power can expressed as $P_{dyn} \propto\!\!\!\sim f^2$. The cores in the little cluster have fewer clock frequency levels and even fewer voltage levels, but the same approximate relationship can be utilized for the regressors representing those cores. Furthermore, for the static portion of the power consumption, the same principle can also be applied. It was estimated to be approximately proportional to $f^{1.5}$. The core utilization is in this scenario expected to be directly proportional to the dynamic power consumption.

Using these approximate relationships as a basis, the polynomials were created as the product of the core utilization to a power of 0 or 1 and the core frequency to a power of between 1 and 3 in increments of 0.5. This was performed for each core and resulted in 58 new nonlinear regressors with a total of 68 regressors, including the original 10.

This approach was implemented in Python 3.7 and utilizes SIPPY, a system identification package developed at University of Pisa, Italy [77].

The N4SID algorithm does not have many parameters that can be tuned. However, the model order can be viewed as a hyperparameter. In this implementation, the selection of nonlinear regressors can also be considered as hyperparameters. Additionally, the data preprocessing step also has to be considered. In this thesis project, however, no preprocessing or filtering, except for resampling, was carried out. The data for this modeling approach was resampled at 5 Hz. A rate that was established using grid search and cross-validation on the 1-hour block length.

Optimization of the utilized regressors was performed using correlation analysis and grid search. A phenomenon that was noticed during the early model selection phase was that using all the regressors led to some overfitting issues. Therefore, to reduce the number of regressors, randomized search cross-validation and correlation analysis was performed. The randomized search was performed with the 1-hour block length for 500 iterations. Each iteration, three random combinations of core frequency to a power between 1 and 3 and core utilization to a power of 0 or 1 were selected. The combinations were then applied to the regressors belonging to each core to create the new regressors. At the end of each iteration, the average MSE was measured. Using the results, a pair-wise correlation analysis was performed to detect each regressor's overall contribution to the error. Figure 6.1 shows that most of the regressors with only a single frequency component showed a positive correlation. That is, they increased the error when they were utilized. Those that showed a negative correlation produced a decrease in the error when they were utilized. The regressors with a positive correlation were therefore removed from the regressor set.

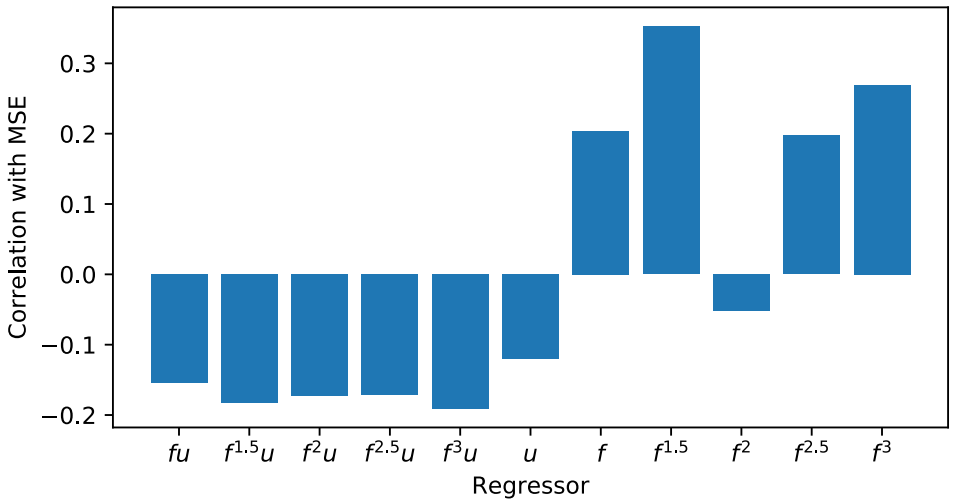

Figure 6.1: Correlation between regressor and MSE.

Grid search and cross-validation were performed as an additional reduction step. During the grid search, the model order was set to 5 for all iterations. This was implemented to reduce computational time. The model order that produces the best performance was, however, expected to be higher than 5. An assumption was made, though, that a fifth-order model would be representative enough for this hyperparameter validation step. All permutations of the remaining regressors were tested and the best regressor configuration was saved. The best regressor set is shown in (6.1), where f is core frequency, u is core utilization and i indicates the number of cores.

$$U_{nl} = [f^{1.5}u_i, f^2u_i, f^3u_i, u_i, f^2], i = 1..8 \tag{6.1}$$

The final number of regressors utilized in this approach is 34. Furthermore, these regressors were selected for implementation for both 1-hour and 6-hour block lengths.

The optimal model order was estimated using the previously established combination of nonlinear regressors. The order of the state-space representation estimated by the N4SID algorithm was optimized using grid search and cross-validation. The average validation error was measured for orders between 2 and 60. Figure 6.2 shows the model performance for each order.

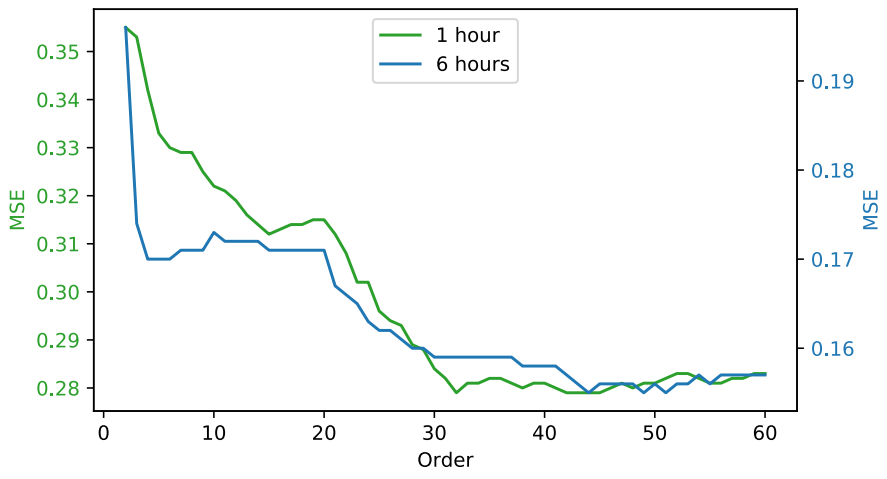

Figure 6.2: Validation error and model order.

The above figure shows that a model of lower order produces the best result for the 1-hour than for the 6-hour block length. The 1-hour model performed the best at 32 while the 6-hour model performed the best at 43.

6.2 Hammerstein-NARX

The second model structure chosen was an NARX approach implemented as an artificial neural network. As shown by Zhang et al. [66] and Sridhar et al. [65], a neural network can be trained to predict the temperature at the next time step based on previous temperature values and some exogenous inputs that affect the temperature. The two approaches were in this implementation combined to create a Hammerstein-NARX structure. In this approach, a network with one hidden nonlinear layer and one linear output layer has been constructed. The inputs are the 10 regressors and their respective values shifted back in time n_x time steps. The nonlinear layer uses a sigmoid activation function to approximate the nonlinearity of the system. The output layer is a linear function that produces a weighted sum of the values that are produced by the nonlinear layer. The output from the linear output layer is fed back to itself for the past n_y time steps.

The idea behind this approach is that the nonlinear static layer can learn to approximate something that is proportional to the power dissipation produced by the system. This is then summed by the dynamic linear output layer to produce a temperature prediction. This approach was implemented in Matlab utilizing the *narxnet* function in the Deep Learning Toolbox. Figure 6.3 shows how the network was structured during training.

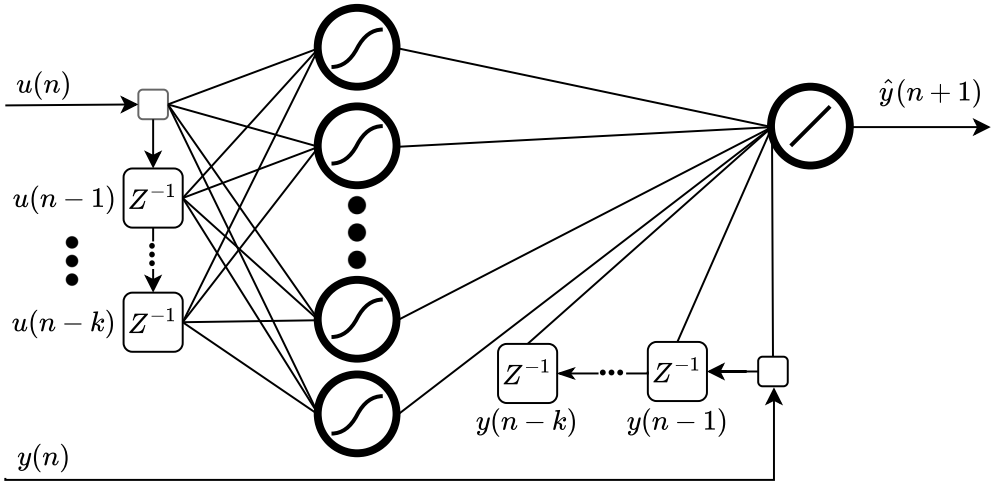

Figure 6.3: Offline Hammerstein-NARX structure used for training.

The network is trained in an offline configuration. This was chosen since an online configuration suffered from the vanishing gradient problem during training. In an offline configuration, there is no recurrence in the network. Thus, the van-

ishing gradient is not an issue. Early stopping on the validation performance was implemented as well. The training was stopped when the error on the validation set started to increase. When the training of the network was finalized, the model structure was closed to produce the online layout shown in Figure 6.4. Using this structure, the network can generate predictions of future values of the temperature without relying on actual temperature measurements as inputs.

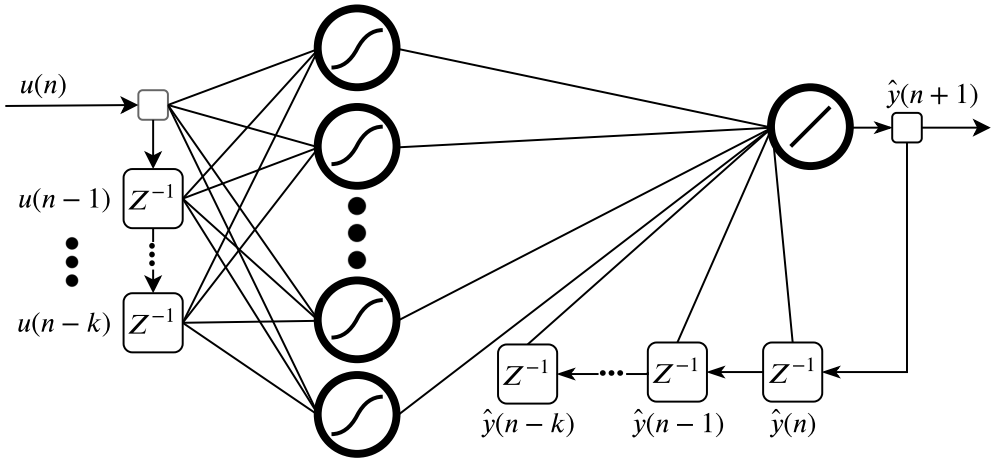

Figure 6.4: Online Hammerstein-NARX structure used to produce predictions.

A few hyperparameters for this approach were selected based on the network structures suggested in [66] and [65], as well as some empirical experience. The activation function was selected to be a sigmoid function. Additionally, only a shallow structure with one hidden layer was tested. The optimization algorithm selected, Levenberg–Marquardt, was also not changed and its associated parameters were kept as the default for the *trainlm* function in Matlab's Deep Learning Toolbox. The Levenberg–Marquardt optimization algorithm was chosen since it was the only algorithm that could successfully converge to a solution during training on the offline configuration. Other optimization algorithms, such as Gradient Descent with Momentum and Bayesian Regularization, failed to converge to a solution or yielded largely varying results when trained several times on the same data.

The hyperparameters that were experimented with for this approach were the number of layers in the hidden layer as well as the number time steps n_x and n_y. The sampling period was also experimented with. All four were optimized using grid search and cross-validation for both the 1-hour and 6-hour implementation.

The number of time steps for the inputs and output, n_x and n_y, produced the lowest error when they were set to 10 and 9, respectively, for both the 1 hour and 6 hours of data. The sampling rate showed a similar performance over a range of

values. Therefore, a sampling frequency of 5 Hz was selected.

In Figure 6.5, the validation performance for different layer sizes is shown. For the 1-hour block length, 3 neurons in the hidden layer produced the best validation performance on average. When training the model structure using 6 hours of data, 5 neurons yielded the lowest average prediction error.

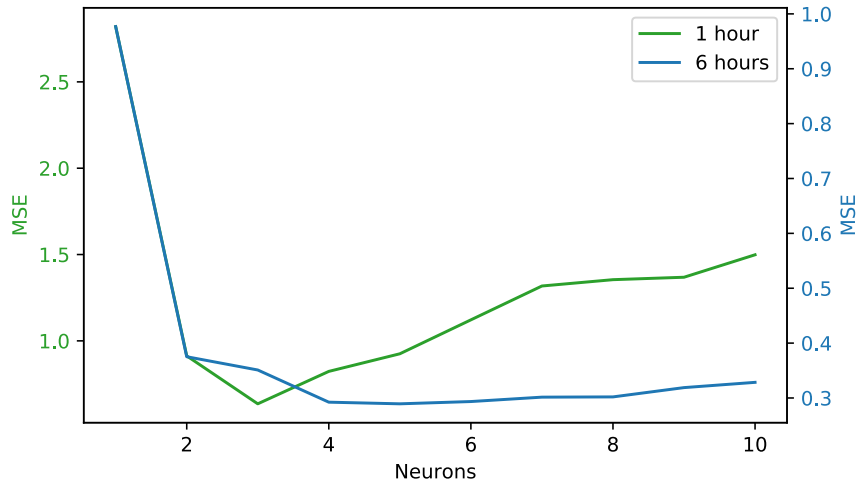

Figure 6.5: Validation error per size of the hidden nonlinear layer.

6.3 FIR-RNN

The final model structure that was assessed is based on a recurrent neural network. This structure has one recurrent layer followed by a single linear layer. This is based on an FIR structure, where the output is predicted solely based on n_x previous inputs.

This modeling approach is based on the structure utilized by Pérez et al. [67]. They found that a shallow structure with either GRU or LSTM layers produced the best performance in their immersive cooling experiment. A similar approach is therefore implemented, as shown in Figure 6.6. A single layer of RNN neurons is followed by a single linear layer. Each time step, the RNN layer takes the input vector and passes it through the neurons to produce a vector of nonlinear states h that is passed to the next time step. This is performed until the current time step is reached. The hidden state vector h is then passed through a linear function to determine the prediction $\hat{y}$. The nonlinear function that is applied inside each

recurrent unit differs depending on whether it is a GRU unit or an LSTM unit and on the activation function that is utilized.

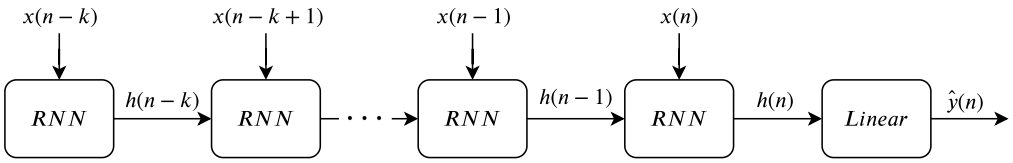

Figure 6.6: FIR-RNN model structure.

This model structure was implemented in Python 3.7 using Keras and Tensorflow. Early stopping on the validation set has also been utilized for this approach.

The hyperparameters that were selected empirically were the optimizer and activation function utilized in the RNN nodes. Pérez et al. [67] utilize the Nesterov Adaptive Momentum (Nadam) optimizer and a *tanh* activation function. Thus, these parameters were selected in this implementation, as well.

The first hyperparameter that was assessed was the number of time steps for the input that had to be considered. Since this approach is not recursive, many time steps have to be included to capture the response of the system. To estimate the settling time of the system, a step response was measured by going from 0 to 100% utilization on all cores when the Odroid board was configured to run at 1800 MHz and 1500 MHz for the big and little cluster, respectively. Figure 6.7 shows that it takes approximately 100 seconds for the system to settle. Therefore, it can be concluded that an FIR model would need the input values for the past 100 seconds to be able to simulate the dynamics of the system accurately.

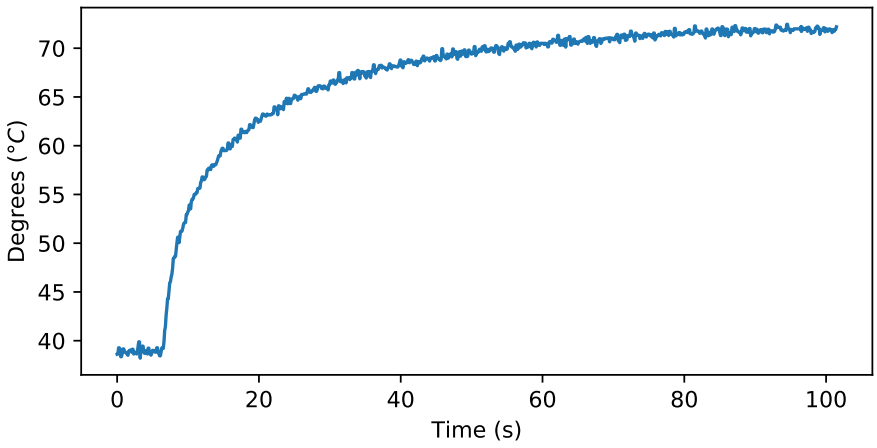

Figure 6.7: Step response of the heterogeneous system.

A settling time of 100 seconds will amount to a different number of time steps depending on which sample rate that is utilized. Furthermore, a uniform sampling rate will cause the input vector to become very large and increase the training time. Therefore, a different sampling scheme was devised. In [59, 60], Li et al. deployed a logarithmic sampling scheme to better capture the effect of the recent time steps and put less emphasis on the not as important earlier time steps. This approach was also implemented in the FIR-RNN model to reduce the order of the model and speed up the training without, hopefully, losing much prediction accuracy.

The sample rate and the number of samples were tested through grid search and cross-validation. Three other hyperparameters were also tested in conjunction: the unit type (LSTM or GRU), the number of units and the batch size. For both the 1-hour and 6-hour block lengths, a sample length of 50 samples spread out logarithmically between 0 and 100 seconds, performed the best. The GRU unit also outperformed the LSTM unit using both block lengths. A batch size of 1 and a unit size of 10 was found to be the optimal values for the 1-hour block length. Using the longer block length, a batch size of 4 and a unit size of 18 generated the lowest average validation error.

7 Evaluation

The previous chapters detailed how the model structures were selected, how the models were trained and how hyperparameters were selected. To validate the performance of the established models, they have to be tested on a previously unseen data set. This chapter features the validation of the models and presents their respective performance on the 2 hours long test set.

7.1 1-hour performance

Using the hyperparameters and model structures described in the previous section, the models were validated through 10-fold cross-validation. Table 7.1 shows the result for the model on the 1-hour block length.

Table 7.1: MSE for the implemented approaches trained with 1 hour of data.

	Folds										
Method	1	2	3	4	5	6	7	8	9	10	**Avg**
Polynomial N4SID	0.16	0.15	0.15	0.16	0.16	0.14	0.16	0.16	0.17	0.14	**0.16**
Hammerstein-NARX	0.53	1.28	0.61	0.74	0.74	0.55	0.65	0.85	0.79	0.54	**0.73**
FIR-RNN	2.28	2.14	1.42	1.44	1.05	2.12	1.60	1.30	2.77	0.80	**1.69**

The Polynomial N4SID approach showed the lowest average MSE. It can also be noted that the N4SID based approach has the by far lowest variance, with a standard deviation of just 0.01. The other two approaches had significantly worse performance on the test data.

Figure 7.1 shows the models' performance on the test set when trained on the seventh fold. This fold is selected since it is the fold that is the closest to the average for all three approaches. The configuration parameters of the board were randomly changed every 10 to 60 seconds. This means that approximately 57 different board configurations were utilized in the 2000 second window shown in Figure 7.1.

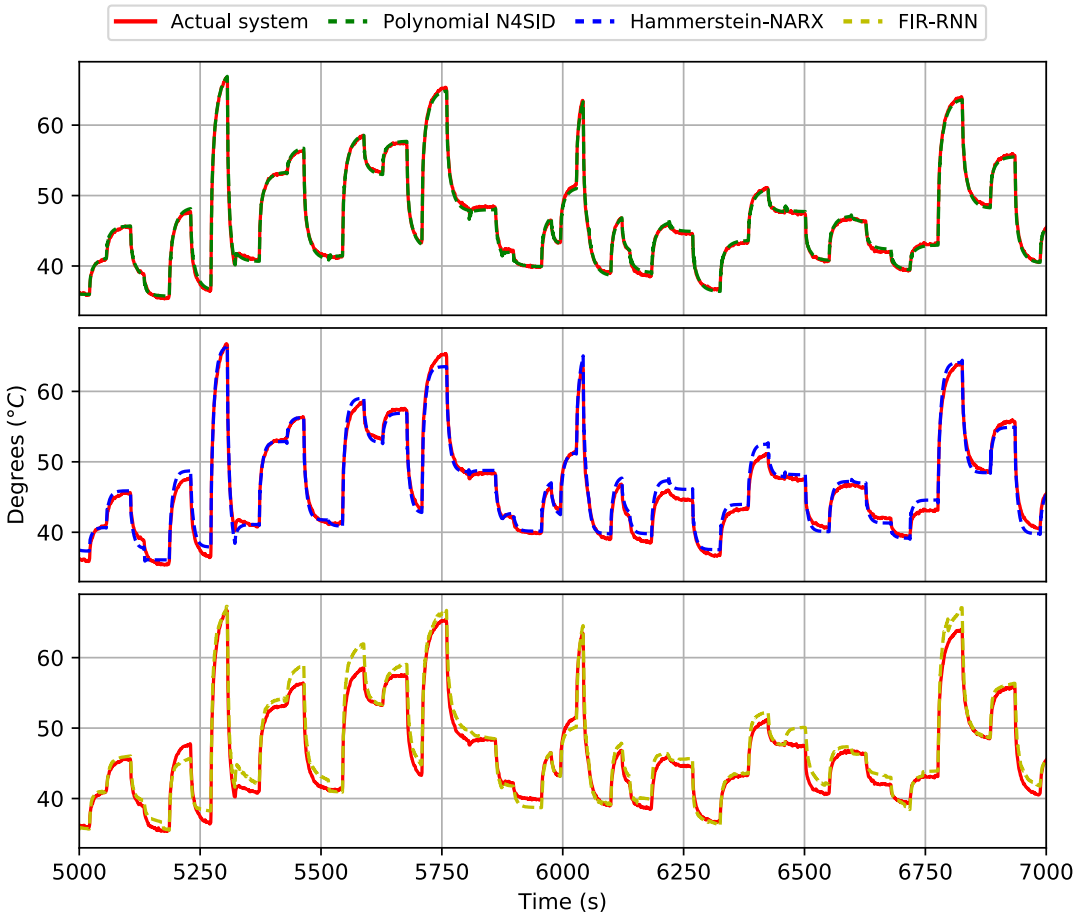

Figure 7.1: 1-hour model predictions on the last 2000 seconds of the test data.

Looking at the above figure, it can be seen that the Polynomial N4SID model produced a good approximation of the true measured temperature. The other two models produced less desirable results, but they still yielded a decent approximation of the true temperature. Furthermore, the Polynomial N4SID model does not appear to have any particular problem areas or specific configurations that it struggles with. The other two models and especially the FIR-RNN show varying performance in regards to the different board configurations.

The average training time, average prediction time and the number of parameters were also measured for the three model structures. Table 7.2 shows that the N4SID-based model structure has the lowest training and prediction time. However, it is closely followed by the Hammerstein-NARX model structure. The FIR-RNN model takes the longest both to train and to make predictions. The training time is especially significant as it is about 100 times that of the other two approaches.

Table 7.2: Average training time, average prediction time and number of parameter for the 1-hour models.

Method	Training time (s)	Prediction time (s)	Number of parameters
Polynomial N4SID	6	0.25	2144
Hammerstein-NARX	7	0.558	347
FIR-RNN	987	4.9	671

7.2 6-hour performance

The same procedure was utilized for the 6-hour block length. The models were validated through 4-fold cross-validation, as detailed in Chapter 5. Table 7.3 shows the result for the model when trained with 6 hours of data.

Table 7.3: MSE for the implemented approaches trained with 6 hours of data.

	Folds				
Method	1	2	3	4	**Avg**
Polynomial N4SID	0.11	0.11	0.11	0.11	**0.11**
Hammerstein-NARX	0.26	0.25	0.28	0.28	**0.27**
FIR-RNN	0.24	0.21	0.18	0.19	**0.21**

The prediction error of the Polynomial N4SID model was reduced even further when trained with 6 hours of data. It improved by approximately 50% compared to its 1-hour performance. The FIR-RNN model, however, has improved substantially. It yields a prediction MSE of 0.21 when trained with more data. The Hammerstein-NARX model did also improve compared to the 1-hour block length, but it did not see the same level of improvement as the recurrent FIR model. Figure 7.2 shows the three modeling approaches' performance on the final 2000 seconds of the test set when trained on the second fold.

The average training time, average prediction time and the number of parameters for the three model structures on 6 hours of training data is shown in Table 7.4. Just as for 1 hour of training data, the N4SID and NARX-based models have significantly lower training and prediction times. Interestingly, the Hammerstein-NARX model's prediction time only increased slightly and is more than twice as fast as the Polynomial N4SID model.

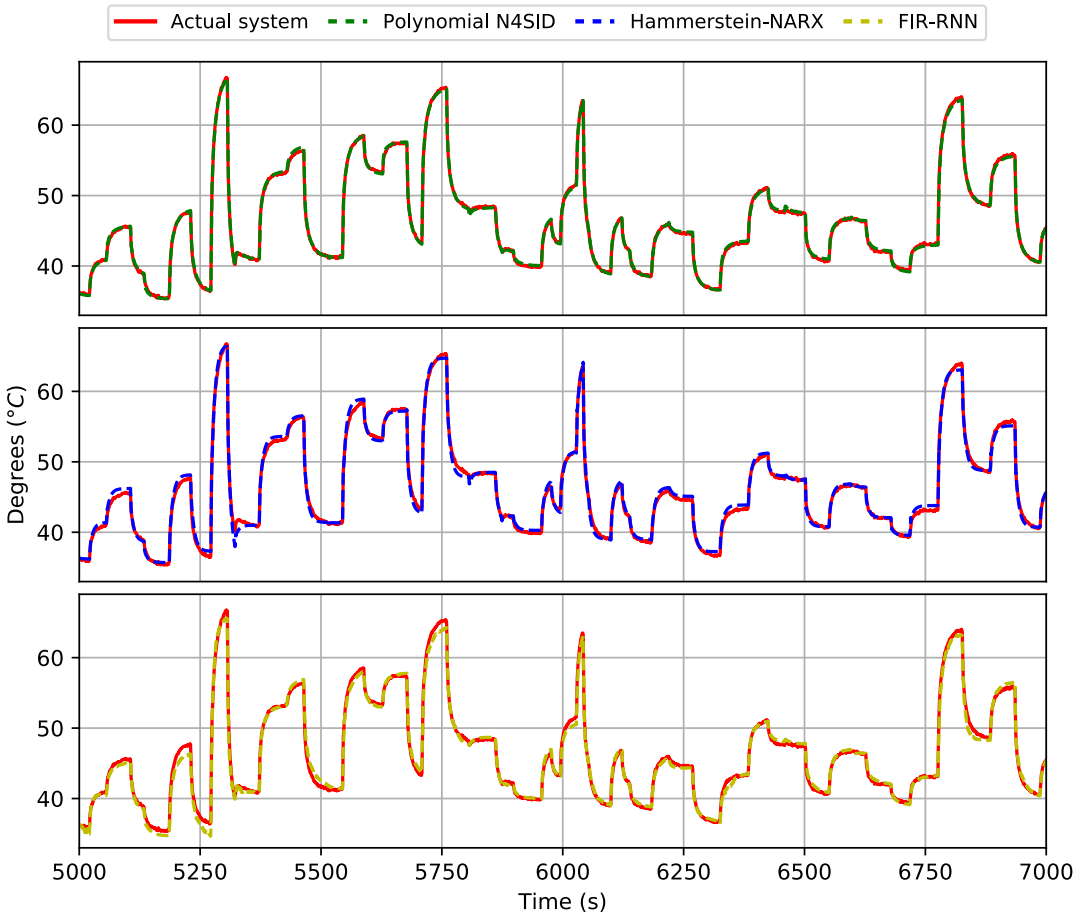

Figure 7.2: 6-hour model predictions on the last 2000 seconds of the test data.

Table 7.4: Average training time, average prediction time and number of parameter for the 1-hour models.

Method	Training time (s)	Prediction time (s)	Number of parameters
Polynomial N4SID	60	1.34	3354
Hammerstein-NARX	67	0.65	571
FIR-RNN	2580	10.5	1639

7.3 Summary

The results presented in the preceding sections reveal that there are several types of modeling approaches that can be utilized to predict the temperature dissipation of a heterogeneous processor. However, one modeling approach outperforms the others in its ability to learn the dynamics of a system from a limited amount of data. The second research question that was posed, aimed at addressing which types of modeling approaches that would be best suited for prediction of heat dissipation with the lowest error. Judging by the results presented in Tables 7.1 and 7.3, it can be concluded that the more model-driven Polynomial N4SID approach performs the best when trained with both 1 hour and 6 hours of data.

8 Discussion

The modeling approaches evaluated in this thesis show that several approaches to thermal modeling of heterogeneous processors can achieve promising results. However, these modeling structures all have their own advantages and limitations. Therefore, this chapter presents an analysis of some potential improvement as well as discussion around why the modeling approaches produced the results that they did.

The Polynomial N4SID model performed the best of the three modeling approaches for both lengths of training data. There are two reasons why it performed so well. The first is that it is constrained to only model linear systems. As the relationship between power and temperature is essentially linear, it is very limited in the way it can fit the model to the data. Therefore, it is less likely to overfit to the training data. The second reason it performed so well is the accuracy of the nonlinear regressors. Since the theoretical formulas for describing the power dissipation in a processor can be easily obtained, the different parts of the power dissipation can be approximated as a set of polynomials.

Even though the state-space-based model structure showed superior performance, there are some caveats to consider. A model of order 43 will yield over 3000 parameters. Furthermore, the training time complexity of the N4SID algorithm is $O(n^2)$ and the prediction time complexity is $O(n)$, where n is the model order. Therefore, it will take significantly longer to train a model of a large order than a model of small order. In this thesis, the objective was to find the model that yielded the highest prediction accuracy. Thus, the training and prediction times were not considered when assessing the performance of the models. However, when deploying a temperature prediction model in a real-world application, especially the prediction time complexity can be crucial. It would, therefore, be better to choose a model of lower order. Luckily, decreasing the order of the model does not affect accuracy too much in this case. In Figure 6.2, it can be seen that a fifth-order model only increases the error with about 25% for the 1-hour model. For 6 hours of data, this difference is only about 10%. Thus, it would be more beneficial to select a lower order model for a real-world implementation.

The Hammerstein-NARX model performed the worst of the modeling approaches when trained on 6 hours of data. This could perhaps be explained by two separate phenomena: the limitations of the nonlinear layer and the network's offline training configuration.

The nonlinear part of the network that is supposed to learn a representation of the power dissipation is perhaps to simple to a more exactly capture the relationship between the regressors and the power dissipation. Since the nonlinear part of the network only comprises a single layer, it will not be able to accurately learn the underlying function since it relies on the product of two or more regressors. A solution to this could be to experiment with deeper network structures and add more nonlinear layers. This will allow the network to learn more complex relationships.

The other issue with the Hammerstein-NARX model structure is that it has to be trained in an offline configuration. This means that it will always have the last measured temperature value as an input. It is, thus, very easy for the network to get good results by just learning to output that value directly. Perhaps a solution to this could be to train the network in a semi-online configuration. In a semi-online configuration, the network is trained to make online predictions for only a limited number of time steps instead of for all time steps. For example, training to predict 10 consecutive temperature values for 10 consecutive configuration points at a time. This would enable the network to trained for a few time steps without relying on the actual temperature measurements too much. It would also suffer less from the vanishing gradient problem than if it had been trained in a completely online configuration.

Even though the Hammerstein-NARX model performed the worst on 6 hours of data, there is still a case to be made why this type of model structure should be considered when deploying a temperature prediction model in a real-world application. As shown in Table 7.4, the Hammerstein-NARX model is more than twice as fast as the Polynomial N4SID model is at making predictions. For applications where fast prediction times and computational efficiency is crucial, this type of model could be very useful. Furthermore, this model should be easier to adapt to different applications. In this thesis, the objective was to model the temperature based on the cluster frequency and core utilization. If other regressors or workload applications also would be considered, this modeling approach would give a lot more flexibility, as it is not based on any theoretical relationship apart from that between power and thermal dissipation.

The FIR-RNN structure performed rather well with 6 hours of training data, though, not as well when trained on a smaller data set. The high complexity of the model is most likely the reason for this difference. The RNN model that was implemented in this thesis constitutes many time steps and has several nodes per time step. The nonlinear neurons have much greater freedom to learn more complex functions than linear neurons do. This can cause the model to overfit more easily if it is trained with a small amount of data. Therefore, it is not very surprising that this type of model structure did require a lot of data to produce low prediction errors. This model structure may produce even better results, given even more data. However, a substantial drawback of this type of model structure is the training time. As shown in Tables 7.4 and 7.2, the training time is between 50 and 120 times longer compared to the other two model structures. This makes this type of model structure very time-consuming to both select hyperparameters for and to validate. Furthermore, the prediction time of this structure was also significantly higher than that of the other model structures. The use case for this type of model structure could be in a scenario where a lot of training data is available, prediction time is not an issue and high prediction accuracy is the priority.

9 Conclusions

In this thesis, system identification approaches for thermal dissipation in processors has been investigated. The goal was to compare the performance of model-driven and data-driven modeling approaches to thermal dissipation in heterogeneous processors. A review of previously proposed approaches to thermal and power dissipation modeling of processors revealed that several approaches had been suggested and successfully applied in previous research. With this as a basis, three parametric system identification approaches were chosen and implemented in a comparative study: a state-space identification method using polynomial regressors and the N4SID algorithm, a Hammerstein-NARX neural network approach and an RNN based modeling approach configured in an FIR configuration. These modeling approaches were each assessed for both 1 and 6 hours of training data collected from a heterogeneous ARM processor and a thermal camera.

The study conducted in this thesis showed that the approach that relied on the state-space system identification algorithm produced the lowest prediction error on both the 1-hour-long and 6-hour-long data sets. The Hammerstein-NARX approach and the FIR-RNN approach did also produce good results when trained with 6 hours of data but did still have approximately twice the prediction error compared to the Polynomial N4SID approach.

All three model structures implemented in this thesis can be considered parametric gray-box approaches that are both model- and data-driven, however, to varying degrees. When trained with only 1 hour of data, the most model-driven approach performed the best, which coheres with the initial assumption that this approach should be the least data-driven. Utilizing 6 hours of training data, the most data-driven approach, the FIR-RNN, showed the biggest increase in performance. This is also not very surprising since providing more training data should naturally allow data-driven models to perform better. The neural network-based Hammerstein-NARX approach that can be considered as equally model- and data-driven, did not quite manage to match the performance of the other model structures on when trained with 6 hours of data. This is likely due to the nonlinear layer not being complex enough to capture the relationship between cluster frequency, core utilization

and power dissipation.

Based on the results of the approaches implemented in this thesis, two main conclusions can be drawn. The first is that it is generally a good idea to identify and break down a model into linear parts if possible. For modeling of heat dissipation in processors, this means that approximating the power dissipation or something that is representative of the power dissipation is advised. In can also be concluded that model-driven modeling should be preferred over data-driven modeling if an approximate theoretical model of a system can be established and computational speed is crucial. A model-driven approach has advantages when it comes to both the training and prediction time complexity as well as the comprehensiveness of the model.

9.1 Future work

This line of research still has some additional obstacles to tackle. There are three main limitations that this thesis project does not take into consideration and that could be expanded upon in further research. The first is utilizing the ambient temperature and humidity as variables. In this project, they were not considered for technical reasons, as no climate-controlled environment was available. For real-world implementations this aspect is crucial and should be considered. Another important and perhaps also more complex aspect to examine in future work is the impact of the workload application. In this thesis project, a static workload application was utilized. Many of the articles presented in Chapter 4, however, suggest methods to represent a workload application as event counters, such as the number of operations of a certain type or the number of cache accesses that are performed. This would allow for a more generally applicable model as it would not be limit to a specific class of applications. A third aspect is multi-output systems. This thesis only considered the maximum temperature across the entire processor SoC, i.e., a MISO system. Predicting the heat dissipation of individual parts of a CPU would allow for a more precise understanding of which parts of a CPU that generate the most heat.

10 Bibliography

[1] S. Buzzi, C. I, T. E. Klein, H. V. Poor, C. Yang, and A. Zappone, "A survey of energy-efficient techniques for 5g networks and challenges ahead," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 697–709, 2016.

[2] C. Preist, D. Schien, and P. Shabajee, "Evaluating sustainable interaction design of digital services: The case of youtube," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland Uk: ACM, 2019, 397:1–397:12.

[3] HSY (Helsingin seudun ympäristöpalvelut), *2018 Climate Actions in the Helsinki Metropolitan Area*. 2018.

[4] G. Association, "The mobile economy 2020," Tech. Rep. [Online]. Available: `https://www.gsma.com/mobileeconomy/`.

[5] A. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, 117–157, 2015. [Online]. Available: `http://dx.doi.org/10.3390/challe6010117`.

[6] M. H. Alsharif, J. Kim, and J. H. Kim, "Green and sustainable cellular base stations: An overview and future research directions," *Energies*, vol. 10, no. 5, p. 587, 2017. [Online]. Available: `http://dx.doi.org/10.3390/en10050587`.

[7] R. Tu, X. Liu, Z. Li, and Y. Jiang, "Energy performance analysis on telecommunication base station," *Lancet*, vol. 43, pp. 315–325, Feb. 2011.

[8] B. Ullman, "Designing an arm-based cloud ran cellular/wireless base station," *embedded.com*, 2013. [Online]. Available: `https://www.embedded.com/designing-an-arm-based-cloud-ran-cellular-wireless-base-station/`.

[9] K. Rupp, "42 years of microprocessor trend data," *https://www.karlrupp.net/*, 2018. [Online]. Available: `https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/`.

[10] P. McGuinness, "What's next for mobile? heterogeneous processing evolves," *www.embedded-computing.com*, 2014. [Online]. Available: `https://www.embedded-computing.com/embedded-computing-design/whats-next-for-mobile-heterogeneous-processing-evolves`.

[11] M. Wolf, "10.4.1 Heterogeneous shared memory multiprocessors," en, p. 40, [Online]. Available: `https://www.sciencedirect.com/topics/computer-science/heterogeneous-multiprocessor`.

[12] G. Peter, "Big.little processing with arm cortex-a15 cortex-a7," ARM Holdings, Tech. Rep., 2011. [Online]. Available: `https://web.archive.org/web/20131017064722/http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf`.

[13] A Janczak, *Identification of nonlinear systems using neural networks and polynomial models : a block-oriented approach*. Springer, 2005.

[14] S. Salehi and R. F. DeMara, "Energy and area analysis of a floating-point unit in 15nm cmos process technology," in *SoutheastCon 2015*, 2015, pp. 1–5.

[15] R. Triggs, "Does moore's law still apply to smartphones in 2020?" *Android Authority*, 2020. [Online]. Available: `https://www.androidauthority.com/moores-law-smartphones-1088760/`.

[16] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, pp. 68 –75, Jan. 2004.

[17] S. R. Kassa and R. K. Nagaria, "A review on robust low power system level digital circuit design approaches in nano-cmos technologies," in *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, ser. ICCCT '15, Allahabad, India: Association for Computing Machinery, 2015, 371–375. [Online]. Available: `https://doi.org/10.1145/2818567.2818676`.

[18] D. Rittman, "Nanometer power leakage," Mar. 2020.

[19] J. De Galas, "The quest for more processing power," 2006. [Online]. Available: `https://www.anandtech.com/show/1611`.

[20] P. Kocanda and A. Kos, "Static and dynamic energy losses vs. temperature in different cmos technologies," in *2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)*, 2015, pp. 446–449.

[21] K. DeVogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "Modeling the temperature bias of power consumption for nanometer-scale cpus in application processors," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, 2014, pp. 172–180.

[22] S. Holmbacka, F. Hällis, W. Lund, S. Lafond, and J. Lilius, "Energy and power management, measurement and analysis for multi-core processors," Tech. Rep. 1117, 2014.

[23] M. Jouaneh, *Fundamentals of Mechatronics*. Cengage Learning, 2012, p. 64. [Online]. Available: `https://books.google.se/books?id=_4y3t_5KO8EC`.

[24] M. Wolf, Ed., *High-Performance Embedded Computing (Second Edition)*. Morgan Kaufmann, 2014, p. 77. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/B9780124105119120015`.

[25] Hardkernel, *Linux kernel for odroid-xu4*, version 4.14.165-172, Jan. 15, 2020. [Online]. Available: `https://github.com/hardkernel/linux/releases/tag/4.14.165-172`.

[26] M. Adams, M. Verosky, M. Zebarjadi, and J. Heremans, "Active peltier coolers based on correlated and magnon-drag metals," *Phys. Rev. Applied*, vol. 11, p. 054 008, 5 2019. [Online]. Available: `https://link.aps.org/doi/10.1103/PhysRevApplied.11.054008`.

[27] J. B. J. Fourier, *The Analytical Theory of Heat*, A. Freeman, Ed., ser. Cambridge Library Collection - Mathematics. Cambridge University Press, 2009.

[28] K. Golicha, "Heat dissipation in a computer," Jan. 2013. [Online]. Available: https://www.researchgate.net/publication/275270691_Heat_Dissipation_in_a_Computer.

[29] A. Moradikazerouni, M. Afrand, J. Alsarraf, S. Wongwises, A. Asadi, and T. K. Nguyen, "Investigation of a computer cpu heat sink under laminar forced convection using a structural stability method," *International Journal of Heat and Mass Transfer*, vol. 134, pp. 1218 – 1226, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0017931018362045.

[30] M. Q. Brewster, *Thermal Radiative Transfer and Properties*. John Wiley Sons, 1992, pp. 56–57.

[31] T. B. Beneventi Bartolini, "An Effective Gray-Box Identification Procedure for Multicore Thermal Modeling," en, *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1097–1110, May 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6381401/ (visited on 02/17/2020).

[32] N. Sathe, "Thermal modeling of many-core processors," Jul. 2010. [Online]. Available: http://hdl.handle.net/1853/34834.

[33] L. Ljung, *System identification : theory for the user*. Prentice Hall PTR, 1999.

[34] G. James, D. Witten, T. Hastie, and R. Tibshirani, Eds., *An introduction to statistical learning: with applications in R*, ser. Springer texts in statistics 103. Springer, 2013.

[35] M. Blachnik, "Reducing time complexity of svm model by lvq data compression," in *Artificial Intelligence and Soft Computing*, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., Springer International Publishing, 2015, pp. 687–695.

[36] Q. Zhang, "Nonlinear system identification with output error model through stabilized simulation," *IFAC Proceedings Volumes*, vol. 37, no. 13, pp. 501 –506, 2004, 6th IFAC Symposium on Nonlinear Control Systems 2004 (NOLCOS 2004), Stuttgart, Germany, 1-3 September, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017312739.

[37] P. P. J. v. d. Bosch and A. C. v. d. Klauw, *Modeling, identification, and simulation of dynamical systems*. CRC Press, 1994, pp. 178–180.

[38] J. H. Williams and Institute of Physics (Gran Bretanya), *Quantifying measurement: the tyranny of numbers*, English. 2016, OCLC: 1139492869. [Online]. Available: https://iopscience.iop.org/book/978-1-6817-4433-9 (visited on 04/04/2020).

[39] I. Jamaludin, A. P. I. D. N. Wahab, S. Khalid, S. Sahlan, Z. Ibrahim, and M. Rahmat, "N4SID and MOESP subspace identification methods," Mar. 2013, pp. 140–145.

[40] P. Van Overschee and B. De Moor, "N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems," *Automatica*, vol. 30, no. 1, pp. 75 –93, 1994, Special issue on statistical signal processing and control. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005109894902305.

[41] R. K. Pearson and M. Gabbouj, *Nonlinear digital filtering with Python: an introduction*. CRC Press, Taylor & Francis Group, 2016, pp. 96–102.

[42] O. Nelles, *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer, 2001.

[43] J. Sjöberg, H. Hjalmarsson, and L. Ljung, "Neural networks in system identification," *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 359 –382, 1994, IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S1474667017477378`.

[44] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, 85–117, 2015. [Online]. Available: `http://dx.doi.org/10.1016/j.neunet.2014.09.003`.

[45] A. Zhang, Z. C. Lipton, L. Mu, and A. J. Smola, *Dive into Deep Learning*. 2020. [Online]. Available: `https://d2l.ai/index.html`.

[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[47] K. Huang, A. Hussain, Q.-F. Wang, and R. Zhang, *Deep Learning*, English. Springer, 2019, pp. 34–38, OCLC: 1088326710. [Online]. Available: `https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=5709961` (visited on 04/16/2020).

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: `1406.1078 [cs.CL]`.

[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. arXiv: `1412.3555 [cs.NE]`.

[51] W. Huang, K. Sankaranarayanan, K. Skadron, R. J. Ribando, and M. R. Stan, "Accurate, pre-rtl temperature-aware design using a parameterized, geometric thermal model," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1277–1288, 2008.

[52] G. Paci, F. Poletti, L. Benini, and P. Marchal, "Exploring temperature-aware design in low-power mpsocs," *International journal of embedded systems*, vol. 3, no. 1-2, pp. 43–51, 2007.

[53] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 463–470.

[54] W. Liu, A. Calimera, A. Nannarelli, E. Macii, and M. Poncino, "On-chip thermal modeling based on spice simulation," in *International Workshop on Power and Timing Modeling, Optimization and Simulation*, Springer, 2009, pp. 66–75.

[55] S. Lee, D. Pandiyan, Jae-sun Seo, P. E. Phelan, and C. Wu, "Thermoelectric-based sustainable self-cooling for fine-grained processor hot spots," in *2016 15th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2016, pp. 847–856.

[56] M. Sadri, A. Bartolini, and L. Benini, "Single-chip cloud computer thermal model," in *2011 17th International Workshop on Thermal Investigations of ICs and Systems (THERMINIC)*, 2011, pp. 1–6.

[57] S. Kuo, C. Pan, P. Huang, C. Fang, S. Hsiau, and T. Chen, "An innovative heterogeneous soc thermal model for smartphone system," in *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2018, pp. 384–391.

[58] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[59] D. Li, S. X. Tan, E. H. Pacheco, and M. Tirumala, "Architecture-level thermal characterization for multicore microprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 10, pp. 1495–1507, 2009.

[60] D. Li, S. Tan, E. Pacheco, and M. Tirumala, "Parameterized architecture-level dynamic thermal models for multicore microprocessors," *ACM Trans. Design Autom. Electr. Syst.*, vol. 15, Feb. 2010.

[61] T. J. A. Eguia, S. X. Tan, R. Shen, E. H. Pacheco, and M. Tirumala, "General behavioral thermal modeling and characterization for multi-core microprocessor design," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, pp. 1136–1141.

[62] Z. Liu, S. X.-D. Tan, H. Wang, Y. Hua, and A. Gupta, "Compact thermal modeling for packaged microprocessor design with practical power maps," *Integration*, vol. 47, no. 1, pp. 71–85, 2014.

[63] R. A. Shetu, T. Toha, M. M. R. Lunar, N. Nurain, and A. B. M. A. Al Islam, "Workload-based prediction of cpu temperature and usage for small-scale distributed systems," in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 01, 2015, pp. 1090–1093.

[64] A. Vincenzi, A. Sridhar, M. Ruggiero, and D. Atienza, "Fast thermal simulation of 2d/3d integrated circuits exploiting neural networks and gpus," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 151–156.

[65] A. Sridhar, A. Vincenzi, M. Ruggiero, and D. Atienza, "Neural network-based thermal simulation of integrated circuits on gpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 23–36, 2012.

[66] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine Learning-Based Temperature Prediction for Runtime Thermal Management Across System Components," en, *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 405–419, Feb. 2018. [Online]. Available: `http://ieeexplore.ieee.org/document/7995115/` (visited on 02/07/2020).

[67] J. Pérez, S. Pérez, J. M. Moya, and P. Arroba, "Thermal prediction for immersion cooling data centers based on recurrent neural networks," in *Intelligent Data Engineering and Automated Learning – IDEAL 2018*, H. Yin, D. Camacho, P. Novais, and A. J. Tallón-Ballesteros, Eds., Cham: Springer International Publishing, 2018, pp. 491–498.

[68] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, 2016.

[69] M. J. Walker, S. Diestelhorst, A. Hansson, D. Balsamo, G. V. Merrett, and B. M. Al-Hashimi, "Thermally-aware composite run-time cpu power models," in *2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2016, pp. 17–24.

[70] M. Walker, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Accurate and stable empirical cpu power modelling for multi- and many-core systems," in *Adaptive Many-Core Architectures and Systems Workshop (15/06/18)*, 2018. [Online]. Available: `https://eprints.soton.ac.uk/421995/`.

[71] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao, "Towards better cpu power management on multicore smartphones," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, ser. HotPower '13, Farmington, Pennsylvania, 2013. [Online]. Available: `https://doi.org/10.1145/2525526.2525849`.

[72] A. Balsini, L. Pannocchi, and T. Cucinotta, "Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures," *ACM SIGBED Review*, vol. 16, pp. 51–56, Nov. 2019.

[73] O. Djedidi, M. Djeziri, N. M'Sirdi, and A. Naamane, "A Novel Easy-to-construct Power Model for Embedded and Mobile Systems - Using Recursive Neural Nets to Estimate Power Consumption of ARM-based Embedded Systems and Mobile Devices," in *15th International Conference on Informatics in Control, Automation and Robotics*, ser. Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics, Porto, Portugal: SCITEPRESS - Science and Technology Publications, Jul. 2018. [Online]. Available: `https://hal-amu.archives-ouvertes.fr/hal-01856579`.

[74] C. King, *Stress-ng*, version 0.10.19, Feb. 7, 2020. [Online]. Available: `http://manpages.ubuntu.com/manpages/bionic/man1/stress-ng.1.html`.

[75] P. Mashkov, T. Pencheva, and B. Gyoch, "Reflow soldering processes development using infrared thermography," in *2009 32nd International Spring Seminar on Electronics Technology*, 2009, pp. 1–6.

[76] T. B. Welch, C. H. G. Wright, and M. G. Morrow, *Real-time digital signal processing from MATLAB to C with the TMS320C6x DSPs*, 2nd ed. CRC Press/Taylor & Francis Group, pp. 55–56.

[77] Giuseppe Armenise, *Sippy*, Feb. 27, 2020. [Online]. Available: `https://github.com/CPCLAB-UNIPI/SIPPY/`.